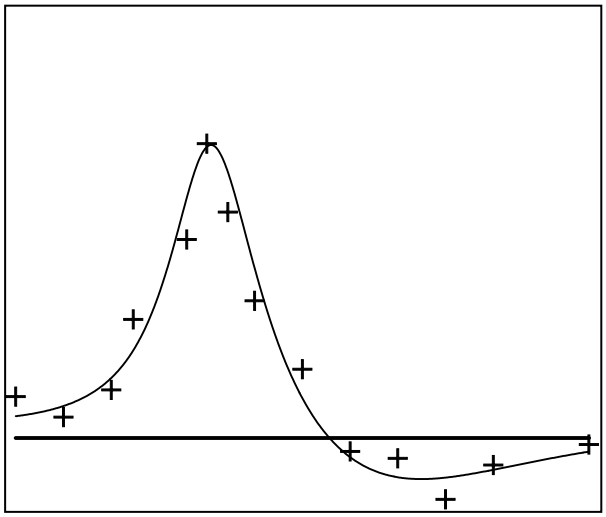
PEST

Model-Independent Parameter Estimation

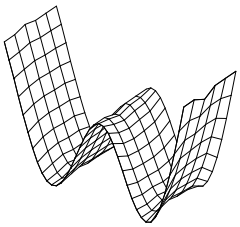
User Manual Part I:

PEST, SENSAN and Global Optimisers

(See part II for documentation of PEST support and uncertainty analysis utilities.)



7th Edition published in 2018
Latest additions: January 2019



Watermark Numerical Computing

Acknowledgements

PEST has written in 1994. Over most of the time between now and then it has been available to the public at no cost. Meanwhile, over all of that period it has been supported, maintained, upgraded and expanded. The reader will no doubt be aware that this has required no small effort from me.

However from time to time, institutions for whom I was consulting, or with whom I have been collaborating, have contributed to its development. Sometimes this was done to allow PEST to better meet their needs. At other times financial help was provided out of pure beneficence of the institution. All of this help has been greatly appreciated.

Accordingly, I would like to acknowledge the help of the following institutions. Without the help of any one of these institutions PEST would be somewhat different from what it is today. Without the help of all of them it would probably not exist today.

- Queensland Department of Environment and Science
- The National Centre for Groundwater Research and Training, Australia
- S. S. Papadopulous and Associates
- Environmental Simulations Inc.
- Aquaveo
- United States Geological Survey
- U.S. Army Corps of Engineers
- Waterloo Hydrogeologic
- CH2MHill
- Department of Civil Engineering, University of Queensland
- University of Idaho
- Idaho National Engineering and Environmental Laboratory
- Los Alamos National Laboratory
- Sandia National Laboratories
- South Florida Water Management District
- St. Johns River Water Management District
- Boise State University
- Matrix Solutions
- Department of Engineering Science, University of Auckland, New Zealand

My deepest apologies are extended to anyone that I have forgotten.

Preface to the Seventh Edition

It is likely that this is the last edition of the PEST manual. It coincides with the release of version 15 of PEST. This, in turn, coinides with the release of version 4 of PEST++.

This is a period of transition. At the end of this period, programs of the PEST++ suite will comprise the preferred packages for inversion, as well as for other model-value-adding tasks that must be performed if modelling is to achieve its true potential in environmental decision support. These include optimisation, uncertainty analysis and optimisation under uncertainty.

However PEST is not about to disappear. It still offers functionality that is offered by no other package. Furthermore, because PEST++ programs also employ a PEST control file (i.e. a PST file), a Jacobian matrix file (i.e. JCO file) and a parameter value file (i.e. a PAR file) whose specifications are the same as those of PEST, utility programs documented in part II of this manual which manipulate and build these types of file can be used to support and expedite usage of programs of the PEST++ suite, as well as those of the PEST suite. (The same applies to programs of the PEST Groundwater Data Utility suite and members of the PEST Surface Water Utility suite.)

Improvements to PEST and its utility suite that have taken place since the release of the sixth edition of parts I and II of the PEST manual include the following.

- PEST and BEOPEST can now be run using the "/f" command-line switch. Through use of this switch, PEST is asked to write a run results file (i.e. a RRF file) in which model outputs based on the contents of a suite of parameter value files are recorded. (Note that PEST_HP also supports this switch.)
- A number of utility programs have been written which manipulate and process the contents of an RRF file. Processing that they perform includes calculation of objective functions based on different PEST control files, and formulation of a randomized Jacobian matrix.
- PEST and many of its utility programs will now tolerate blank lines, comments and "++" lines in a PEST control file. This facilitates inter-operability with the PEST++ suite of programs.
- A utility program named PSTCLEAN removes blank lines, comments and "++" lines from a PEST control file so that a file which includes these features can be read by those of PEST's utility support programs which do not tolerate them.

John Doherty

June, 2018

Preface to the Sixth Edition

It has now been over 10 years since publication of the fifth edition of the PEST manual. Over this time, improvements to PEST and expansion of its utility suite have continued apace. Each new item of functionality was documented in the addendum to the fifth edition of the PEST manual as it was completed. As the number of new utilities expanded, and as PEST itself was enhanced, the addendum grew larger. As this continued, the task of providing a sixth edition of the PEST manual in which all documentation was rationalized, re-ordered and re-formatted grew more and more daunting. So it was postponed, and postponed again, as my efforts were devoted to improving the suite of PEST software rather than rationalizing its documentation.

The many features that were added to PEST and its supporting utility software over this time include

- better tolerance of bad model numerical behaviour which can potentially incur a deterioration in the quality of finite-difference derivatives;
- expanded parallel run management, including the addition of BEOPEST to the PEST suite (BEOPEST employs TCP/IP for communication between master and slaves);
- a suite of utility programs which undertake linear parameter and predictive, pre- and post-calibration error and uncertainty analysis;
- utility software that calculates parameter identifiability, and that employs singular value decomposition to decompose parameters and model outputs into so-called "super parameters" and "super observations";
- linear analysis software which examines the effects of model defects on calibration-induced parameter/predictive bias;
- nonlinear uncertainty analysis implemented using the null space Monte Carlo methodology;
- a new mode of PEST operation known as "pareto" mode which implements direct predictive hypothesis-testing and computer-aided manual regularisation;
- an enhanced ability to query, manipulate and process data contained in binary Jacobian matrix files;
- utilities which support model-independent parameter preprocessing and observation postprocessing;
- utilities that undertake parameter-estimation-relevant operations on matrices;
- two PEST-compatible global optimisers.

Thirty-two bit and sixty-four bit versions of many of these programs are available. The source code for PEST and all of its utility suite is also available; hence users can compile their own version of the entire PEST suite for use on a UNIX platform.

With publication of this sixth edition of the PEST manual, the long period of procrastination is over. This edition of the PEST manual provides documentation for PEST and all of its support utilities in a much more coherent form than was available through the fifth edition of the PEST manual in conjunction with its extensive addendum. Because of the vastly expanded capabilities provided by PEST and its utility support software, this manual is now published in two parts. The first part covers PEST itself, together with the PEST-compatible SCEUA_P and CMAES_P global optimisers, as well as the basic SENSAN sensitivity analyser. The second part of the manual documents PEST's utility support software.

There is actually a third part to the PEST manual. This is a book named "Calibration and Uncertainty Analysis for Complex Environmental Models" published in 2015 by Watermark Numerical Computing and written by myself, John Doherty. It describes in detail the theory of parameter estimation and uncertainty analysis in general, and that which is implemented by PEST in particular; it also suggests ways in which this theory should be translated into best history-matching and uncertainty analysis practice, particularly in contexts where models are built to support environmental management and decision-making. Version 6 of the PEST manual refers extensively to this book. This reduces the length of the manual, as there is no longer a requirement for the manual to document the theory on which PEST and its utility support software rests. The book can be downloaded from the PEST web pages at

http://www.pesthomepage.org

As many readers are aware, the utility programs that are documented in part II of this manual are not the only programs that support the use of PEST in environmental model calibration and uncertainty analysis. Other extensive suites of utility software are available that expedite the use of PEST with groundwater and surface water models. See, in particular, the PLPROC "parameter list processor" and the TSPROC "time series processor". These expanded suites of utility software have their own documentation. Together with their respective manuals, they can be downloaded from the PEST web pages.

John Doherty

April, 2016

Preface to the Fifth Edition

Since publication of the fourth edition of the PEST manual, major improvements have been made to PEST. All of these have been made with the intention of improving PEST's performance in the calibration of large complex models, where many parameters require estimation through a regularised inversion process.

The most profound advance is the "SVD-assist" scheme. This method combines two important regularisation methodologies, namely truncated singular value decomposition and Tikhonov regularisation. The result is a methodology that has the numerical stability of the former scheme and the "parameter friendliness" of the latter scheme. What is more, however, is that regularised inversion can now be carried out with stunning gains in model run efficiency. This is because the number of model runs required per optimisation iteration no longer needs to equal or exceed the number of parameters being estimated; in fact, in some cases, it can be as few as one tenth the number of estimable parameters. The repercussions of this are profound, for it makes regularised inversion easy with models for which it would previously have been considered impossible.

Other PEST improvements include the following.

- Regularisation constraints can now be subdivided into separate groups in order to facilitate the application of these constraints in ratios that are best tuned to the current problem.
- Singular value decomposition is offered as an inverse problem solution device.
- As an alternative to singular value decomposition, "automatic user intervention" can be employed to reduce the dimensionality of a parameter estimation problem in order to prevent the onset of numerical instability where data is insufficient for the simultaneous estimation of all parameters.
- PEST's parallelisation functionality has undergone considerable improvement. Slaves can now enter the parallelisation process late. Furthermore a special restart function has been provided for Parallel PEST, allowing it to restart at the very run where its execution was previously interrupted.
- PEST writes a number of new output files as it undertakes the parameter estimation process. The condition number of the parameter estimation problem is now available at any time, as are residuals pertaining to the latest upgraded parameter set. If singular value decomposition is employed, information on singular values and (optionally) parameter eigenvalues is also available at any time.
- The Jacobian and other matrices can now be stored in compressed form. This allows greater memory and runtime efficiency when estimating parameters *en masse* in highly regularised settings.
- A number of new utility programs have been added to the PEST suite to enhance manipulation of its input dataset and to aid in postprocessing of PEST output.

As time goes on, the simulation of environmental processes as an aid to the management of natural resources is becoming more and more commonplace. If a model is to be used to predict future environmental behaviour, that model must be capable of replicating historical behaviour of that system. Hence it must be calibrated. Where a system is complex (as all

environmental systems are), then tens, hundreds or even thousands of parameters may require estimation. In many cases these parameters cannot be uniquely estimated. Attempts are thus often made to simplify the representation of environmental processes by a model, reducing the number of parameters that require estimation.

Regularised inversion recognises the fact that the number of parameters that can be estimated during the model calibration process is limited by the information content of the dataset available for calibration. However it does not rely on inappropriate or "artificial" methods of model simplification as the price to be paid for numerical stability of the inversion process. Instead, the philosophy that underpins regularised inversion is "let the inversion process itself determine the dimensionality of the parameter estimation problem, in accordance with the information content of the data". Hence simplification is undertaken in a manner that allows the estimation of as many different *combinations* of parameters as the data will allow, while imposing smoothness, equality, or other constraints on those combinations of parameters which cannot be estimated. Because a user rarely knows in advance the optimal simplification strategy for a given calibration context, the use of many parameters gives the calibration process the freedom that it needs to simplify model parameterisation in a manner that is perfectly tuned to that context. Hence the use of many parameters provides the means of extracting maximum information content from a given dataset (which is often very expensive to acquire), and of allowing as much of this information as possible to be reflected in the parameterisation of a model used to simulate salient environmental processes. However, in doing this, the unwanted effects of "overfitting" are prevented through the imposition of a suitable "fitting limit", this being set at a level that is appropriate to the measurement and "structural" noise content of the calibration dataset.

PEST is by far the most advanced parameter estimation package available to environmental modellers. Yet its development is still continuing apace. In recognition of the fact that regularised inversion allows the user to include in his/her model that level of system complexity that *must* exist to explain the data, but that this level of complexity may fall well short of that which actually *does* exist, continued PEST development is now being targeted at "filling in the parameter gaps". This can only be done in a probabilistic sense. However the cost of ignoring such complexity may be the introduction of substantial bias in critical model predictions. Hence further PEST development is now being aimed at the assessment of uncertainty in model predictions as it depends not just on measurement and structural noise (as is presently handled using PEST's predictive analyser) but also on the probabilistic behaviour of system complexity that *may* exist, but simply cannot be captured by the calibration process.

I personally have found the development of PEST to be a truly exiting endeavour, allowing modellers to make better use of their sophisticated and ingenious models as a mechanism for interpreting complex environmental data, with the aim of managing the environment better. I hope that as more model users also become PEST users, they share my excitement in the opportunities available to them for wringing every ounce of information out of hard-won data, thereby allowing them to make important environmental decisions in a more enlightened manner than would otherwise be possible.

John Doherty

July, 2004

Preface to the Fourth Edition

Production of the fourth edition of the PEST manual marks two important milestones in the development of PEST. The first of these is the addition of advanced and powerful regularisation functionality underpinning the release of version 5.0 of PEST. The second is PEST's change in status from that of a commercial product to that of a public domain package.

Over the last few years, the continued development of PEST has focussed on its ability to work successfully with complex, highly-parameterised models. First there was PEST's user-intervention functionality, this allowing the user to hold troublesome parameters (normally insensitive and/or highly correlated parameters) at their current values so that the parameter estimation process could proceed without the damaging effects that these parameters have on that process. The second was the incorporation of PEST's nonlinear predictive analysis functionality, this denoting a recognition of the fact that increased parameterisation normally results in increased parameter nonuniqueness at the same time as any semblance of model linearity rapidly fades from view. Now, with version 5.0 of PEST, comes the advent of advanced regularisation functionality. At the time of writing this preface, PEST's new regularisation functionality has already proven itself enormously useful in the parameterisation of heterogeneous two- and three-dimensional spatial model domains, especially when accompanied by the use of flexible methods of spatial parameter definition such as "pilot points" (see the PEST Groundwater Utility Suite). Use of PEST in "regularisation" mode allows the modeller to estimate many more parameters than would otherwise be possible. Thus, when working with spatial models, PEST is able to "find for itself" regions of anomalous physical or hydraulic properties rather than requiring that such areas be delineated in advance by the modeller using zones of piecewise parameter constancy. Furthermore, the process is numerically very stable, avoiding the deleterious effects on this process of parameter insensitivity or excessive correlation that often accompanies an attempt to estimate too many parameters.

The decision to place PEST in the public domain was not taken lightly. However two factors made the decision almost impossible to avoid. One of these was the advent of competing, public domain software which, while not having anything like the functionality of PEST, is nevertheless highly visible and has US government auspice. The other consideration was of a more philanthropic nature. My instincts tell me that the biggest issue in environmental modelling over the next decade will be that of predictive uncertainty analysis. PEST has a substantial contribution to make in this regard. It is my hope that by making PEST freely available to all modellers at zero cost, it will make an even more important contribution to environmental management based on computer simulation of real-world systems than it already has to date.

Other new features found in version 5 of PEST that were not available in previous versions of PEST include the following.

- All names pertaining to parameters, parameter groups, observations, observation groups and prior information items can now be up to twelve characters in length.
- Prior information items must now be assigned to observation groups.
- Uncertainties in observations and prior information equations used in the inversion process can now be expressed in terms of covariance matrices, rather than simply in terms

of weights.

- If derivatives of model outputs with respect to adjustable parameters can be calculated by the model, rather than by PEST through the use of finite differences, then PEST can use these derivatives if they are supplied to it through a file written by the model.
- Different commands can be used to run the model for different purposes for which the model is used by PEST (namely testing parameter upgrades, calculating derivatives with respect to different parameters, etc.).
- PEST can now send "messages" to a model, allowing the model to adjust certain aspects of its behaviour depending on the purpose for which it is run by PEST.
- PEST stores the Jacobian matrix corresponding to the best set of parameters achieved up to any stage of the parameter estimation process in a special binary file which is updated as the parameter estimation process progresses. A new utility program named JACWRIT re-writes the Jacobian matrix in text format for user-inspection.
- PEST prints out a more comprehensive suite of information on composite parameter sensitivities than was available in previous versions.
- A new utility named PAR2PAR has been added to the PEST suite. This is a "parameter preprocessor" which allows the user to manipulate parameters according to mathematical equations of arbitrary complexity before these parameters are supplied to the model.

John Doherty
January, 2002

Preface to the Third Edition

Production of the third edition of the PEST manual coincides with the release of Version 4.0 of PEST. The principal addition to PEST functionality encapsulated in version 4.0 of PEST is the provision of predictive analysis capabilities to complement PEST's existing parameter estimation capabilities.

With the increasing use of nonlinear parameter estimation techniques in model calibration, there is a growing realisation among modellers of the extent of nonuniqueness associated with parameter values derived through the model calibration process. This realisation is accompanied by a growing desire to examine the effect of parameter nonuniqueness on the uncertainty of predictions made by calibrated models. The importance of quantifying predictive uncertainty cannot be understated. It can be argued that model parameters are often something of an abstraction, sometimes bearing only a passing resemblance to quantities that can be measured or even observed in real-world systems. However the same is not true of model predictions, for these are the reason why the model was built in the first place. If model predictions are being relied upon to serve as a basis for sound environmental management (as they often are), then an ability to quantify the uncertainty associated with such predictions is as important as the ability to make such predictions in the first place.

The concept of a "prediction" can be broadened to describe PEST's use in those fields where parameter estimation is an end in itself. This is especially the case in the geophysical context where PEST is used to infer earth properties from measurements made at the surface and/or down a number of boreholes. In this case it would appear that model parameters (as determined through the nonlinear parameter estimation process) are of overriding importance, and indeed that the concept of a "model prediction" is inapplicable. However this is not the case; in fact PEST's predictive analysis capabilities have proved an extremely useful addition to the exploration geophysicist's arsenal. Use of PEST in "predictive analysis" mode allows the geophysical data interpreter to ask (and have answered) such questions as "is it possible that a hole drilled at a certain location will not intersect any conductive material?", or "what is the maximum possible depth extent of the conductor giving rise to anomalous surficial measurements?"

The term "predictive analysis" as used in this manual describes the task of calculating the effect of parameter uncertainty, as estimated through the calibration process, on predictive uncertainty. A number of methods have been documented for undertaking such an analysis, for example Monte-Carlo methods and linear uncertainty propagation. However unlike many other methods, the PEST predictive analysis algorithm relies on no assumptions concerning the linearity of a model; furthermore, notwithstanding the fact that calculation of model predictive uncertainty is a numerically laborious procedure, PEST's predictive analysis algorithm is less numerically intensive than many other methods of nonlinear predictive uncertainty analysis.

It is hoped that the use of PEST's predictive analyser will allow modellers from all fields of science and engineering to make yet another quantum leap in the productive use of computer simulation models in whatever field of study they are currently engaged.

John Doherty
October, 1999

Preface to Second Edition

Since the first version of PEST was released in early 1994 it has been used all over the world by scientists and engineers working in many different fields, including biology, geophysics, geotechnical, mechanical, aeronautical and chemical engineering, ground and surface water hydrology and other fields. Through the use of PEST in model calibration and data interpretation, many PEST users have been able to use their models to much greater advantage than was possible when such tasks were attempted manually by trial and error methods.

This second edition of the PEST manual coincides with the release of version 3.5 of PEST. Some of the enhancements that were included in this new PEST have arisen out of my own experience in the application of PEST to the calibration of large and complex models. Others have been included at the suggestion of various PEST users, some of whom are applying PEST in unique and interesting situations. For those already familiar with PEST a brief summary of new features follows.

A version of PEST called "Parallel PEST" has been created. This allows PEST to run a model on different machines across a PC network, thereby reducing overall optimisation time enormously.

By popular demand, parameter, observation, parameter group and prior information names can now be up to 8 characters in length. The previous limit of 4 characters per name was set as a memory conservation strategy, a matter of diminishing concern as computing hardware continues to improve.

Observations can now be collected into groups, with the contribution made to the objective function by each group reported through the optimisation process. This information is extremely helpful in the assignment of weights to different measurement types.

PEST no longer ceases execution with an error message if a parameter has no effect on observations; rather it simply holds the offending parameter at its initial value.

PEST can be asked to run a model only once and then terminate execution. In this way PEST can be used simply for objective function calculation. Alternatively, it can be asked to run the model only as many times as is necessary in order to calculate the parameter covariance matrix and related statistics based on initial parameter estimates.

Two new programs have been added to the PEST suite. These are SENSAN, a model-independent sensitivity analyser, and PARREP, a utility that facilitates the commencement of a new PEST run using parameter values generated on a previous PEST run.

However by far the most important changes to PEST are the improved capabilities that it offers for user intervention in the parameter estimation process. Every time that it calculates the Jacobian matrix, PEST now stores it on file for possible later use. It records on another file the overall sensitivity of each parameter, this being related to the magnitude of the vector comprising the column of the Jacobian matrix pertaining to that parameter. Thus, at any stage of the optimisation process, sensitive and insensitive parameters can be distinguished. It is the insensitive parameters that can cause most problems in the calibration of complex models (especially where parameters are many and correlation is high).

At any stage of the optimisation process a user can request that certain, troublesome, parameters be held at their current values. Such parameters can be demarcated either

individually, or according to whether their sensitivity drops below a certain threshold in the course of the parameter estimation process. Alternatively, a user can request that the *N* least sensitive parameters within a certain group be held while the parameter upgrade vector is calculated, where *N* is supplied by the user according to his/her knowledge and intuition with regard to the current parameter estimation problem. As well as this, certain variables controlling how the parameter upgrade vector is calculated can now be altered during a PEST run.

Calculation of the parameter upgrade vector can now be repeated. Thus if a user thinks that PEST could have done a better job of lowering the objective function during a certain optimisation iteration, he/she can halt PEST execution, instruct PEST to hold certain parameters at current values, and ask PEST to calculate the parameter upgrade vector again. This can be done without the need to re-calculate the Jacobian matrix (the most time-consuming part of PEST's operations) because the latter is stored every time it is calculated in anticipation of just such a request.

As an aid to identification of recalcitrant parameters, PEST now records the parameters that underwent maximum factor and relative changes during any parameter upgrade event, these often being the parameters that create problems.

It is important to note that even though PEST has changed somewhat and includes a number of new and powerful features, file protocols used with previous versions of PEST are identical to those used by the latest version of PEST, with one exception; this is the addition of observation group data to the PEST control file. However the new version of PEST is able to recognise a PEST control file written for an older PEST version, and will read it without complaint, assigning a dummy group name to all observations.

PEST has stood the test of time. When it was initially released it offered entirely new possibilities for model calibration and data interpretation. Slowly but surely the PEST user base is expanding as more and more scientists and engineers are realising the benefits that can be gained through the use of these possibilities. The latest version of PEST, which includes Parallel PEST and the options for user intervention briefly outlined above, allows the use of PEST to be extended to the calibration of large and complex models to which the application of nonlinear parameter estimation techniques would have hitherto been considered impossible. It is hoped that new and existing PEST users can apply PEST to new and exciting problems as a result of these enhancements, and that they will be able to harness the potential for more sophisticated and efficient use of models than ever before.

John Doherty
October, 1998

Preface to First Edition

This document describes the use of PEST, a model-independent parameter estimator.

Nonlinear parameter estimation is not new. Many books and papers have been devoted to the subject; subroutines are available in many of the well-known mathematical subroutine libraries; many modelling packages from all fields of science include parameter estimation as a processing option; most statistical and data-analysis packages allow curve-fitting to a user-supplied data set. However in order to take advantage of the nonlinear parameter estimation facilities offered by this software, you must either undertake a modelling task specific to a particular package, you must alter the source code of your model so that it meets the interface requirements of a certain optimisation subroutine, or you must re-cast your modelling problem into a language specific to the package that you are using.

While PEST has some similarities to existing nonlinear parameter estimation software (it uses a powerful, yet robust, estimation technique that has been extensively tested on a wide range of problem types), it has been designed according to a very different philosophy. What is new about PEST is that it allows you to undertake parameter estimation and/or data interpretation using a particular model, without the necessity of having to make any changes to that model at all. Thus PEST adapts to an existing model, you don't need to adapt your model to PEST. By wrapping PEST around your model, you can turn it into a non-linear parameter estimator or sophisticated data interpretation package for the system which your model simulates. The model can be simple or complex, home-made or bought, and written in any programming language.

As far as I know, PEST is unique. Because of its versatility and its ability to meet the modeller “where he or she is at”, rather than requiring the modeller to reformulate his/her problem to suit the parameter estimation process, I believe that PEST will place the nonlinear parameter estimation method into the hands of a wider range of people than has hitherto been possible, and will allow its application to a wider range of problem types than ever before. I sincerely hope that this will result in a significant enhancement in the use of computer modelling in understanding processes and interpreting data in many fields of study.

However you should be aware that nonlinear parameter estimation can be as much of an art as it is a science. PEST, or any other parameter estimator, can only be used to complement your own efforts in understanding a system and inferring its parameters. It cannot act as a substitute for discernment; it cannot extract more information from a dataset than the inherent information content of that dataset. Furthermore, PEST will work differently with different models. There are many adjustments which you can make to PEST to tune it to a specific model, and you need to know what these adjustments are; often it is only by trial and error that you can determine what are its best settings for a particular case. The fact that PEST’s operation can be tuned in this manner is one of its strengths; however this strength can be properly harnessed only if you are aware of what options are available to you.

So I urge you to take the time to understand the contents of this manual before using PEST to interpret real-world data. In this way you will maximise your chances of using PEST successfully. Experience has shown that for some difficult or “messy” models, the setting of a single control variable can make the difference between PEST working for that model or not. Once the correct settings have been determined, PEST can then be used with that model forevermore, maybe saving you days, perhaps weeks, of model calibration time for each new problem to which that model is applied. Hence a small time investment in understanding the

contents of this manual could yield excellent returns.

So “good luck” in your use of PEST; I hope that it provides a quantum leap in your ability to calibrate models and interpret field and laboratory data.

John Doherty
February, 1994

Disclaimer

The user of this software accepts and uses it at his/her own risk.

The author does not make any expressed or implied warranty of any kind with regard to this software. Nor shall the author be liable for incidental or consequential damages with or arising out of the furnishing, use or performance of this software.

PEST Control Variables

See appendix A of part I of this manual for a complete specification of the PEST control file, including a list of all PEST input variables and a description of their roles.

Bugs

If you discover a bug in PEST or any of its utilities, please report it to me, John Doherty, at the following email address.

pestsupport@ozemail.com.au

Table of Contents

1. Introduction....1
1.1 Installation....1
1.2 Software....1
1.3 Concepts....2
1.4 Philosophy....3
1.5 PEST- The Book....4
1.6 Some Practical Considerations....4
1.6.1 The Model....4
1.6.2 Model Input and Output Files....5
1.6.3 WINDOWS and UNIX....6
1.6.4 Differentiability....6
1.6.5 Local Optima....7
1.6.6 Observations and Predictions....7
1.6.7 Numerical Burden....8
1.7 Some Common Tasks....8
1.7.1 General....8
1.7.2 Over-Determined Model Calibration....8
1.7.3 Highly Parameterized Inversion....9
1.7.4 Parameter Preprocessing and Observation Postprocessing....9
1.7.5 Linear Sensitivity and Uncertainty Analysis....9
1.7.6 Jacobian and General Matrix Manipulation....10
1.7.7 Testing the Integrity of Finite-Difference Derivatives....10
1.7.8 Nonlinear Uncertainty Analysis....10
1.7.9 In Conclusion....11
2. The Model-PEST Interface....12
2.1 PEST Input Files....12
2.2 Template Files....12
2.2.1 Model Input Files....12
2.2.2 An Example....13
2.2.3 The Parameter Delimiter....14
2.2.4 Parameter Names....15
2.2.5 Setting the Parameter Space Width....15
2.2.6 How PEST Fills a Parameter Space with a Number....17
2.2.7 Multiple Occurrences of the Same Parameter....19
2.2.8 Preparing a Template File....20
2.3 Instruction Files....20
2.3.1 Precision in Model Output Files....20
2.3.2 How PEST Reads a Model Output File....21
2.3.3 An Example Instruction File....21
2.3.4 The Marker Delimiter....22
2.3.5 Observation Names....23
2.3.6 The Instruction Set....23
2.3.7 Making an Instruction File....34

3. What PEST Does 35
3.1 General 35
3.2 The PEST Control File 35
3.2.1 General 35
3.2.2 Sections of the PEST Control File 35
3.2.3 Naming Conventions 37
3.3 Modes of Operation 37
3.3.1 General 37
3.3.2 Estimation Mode 38
3.3.3 Regularisation Mode 39
3.3.4 Predictive Analysis Mode 39
3.3.5 Pareto Mode 40
3.3.6 Another "Mode" 40
3.4 Parameter Adjustment 40
3.4.1 General 40
3.4.2 Parameter Transformation 41
3.4.3 Fixed and Tied Parameters 41
3.4.4 Upper and Lower Parameter Bounds 41
3.4.5 Optional Alternative Bounds Accommodation 42
3.4.6 Scale and Offset 43
3.4.7 Global Parameter Scaling 44
3.4.8 Parameter Change Limits 44
3.4.9 Damping of Parameter Changes 46
3.5 The Calculation of Derivatives 47
3.5.1 General 47
3.5.2 Forward, Central and Five-Point Differences 48
3.5.3 Parameter Increments for Two and Three-Point Derivatives 50
3.5.4 Settings for Higher Order Derivatives 51
3.5.5 How to Obtain Derivatives You Can Trust 52
3.5.6 Looking at Model Outputs under the Magnifying Glass 54
3.5.7 Split Slope Analysis 55
3.5.8 Model-Calculated Derivatives 57
3.5.9 Using a Surrogate Model for Derivatives Calculation 57
3.6 The Jacobian Matrix File 57
3.7 The Objective Function 58
3.7.1 Weights 58
3.7.2 Covariance Matrices 59
4. The PEST Control File 61
4.1 Introduction 61
4.1.1 General 61
4.1.2 The Role of the PEST Control File 61
4.1.3 Some Specifications 61
4.1.4 Comments, Blank Lines and "++" Lines 63
4.2 Control Data Section 64
4.2.1 General 64
4.2.2 First Line 64

4.2.3 Second Line....64
4.2.4 Third Line....64
4.2.5 Fourth Line....66
4.2.6 Fifth Line....67
4.2.7 Sixth Line....71
4.2.8 Seventh Line....74
4.2.9 Eighth Line....78
4.2.10 Ninth Line....81
4.3 Sensitivity Reuse Section....83
4.4 Singular Value Decomposition Section....83
4.4.1 General....83
4.4.2 Second Line....84
4.4.3 Third Line....84
4.4.4 Fourth Line....85
4.5 LSQR Section....85
4.5.1 Second Line....86
4.5.2 Third Line....86
4.5.3 Fourth Line....87
4.5.4 Some Notes....87
4.6 Automatic User Intervention Section....88
4.7 SVD Assist Section....88
4.8 Parameter Groups Section....89
4.8.1 General....89
4.8.2 Parameter Group Variables....89
4.9 Parameter Data Section....92
4.9.1 General....92
4.9.2 First Part....93
4.9.3 Second Part....96
4.10 Observation Groups Section....96
4.11 Observation Data Section....97
4.12 Derivatives Command Line Section....98
4.13 Model Command Line Section....98
4.14 Model Input/Output Section....101
4.15 Prior Information Section....102
4.16 Predictive Analysis Section....104
4.17 Regularisation Section....104
4.18 Pareto Section....104
4.19 Covariance Matrix Files....105
4.19.1 General....105
4.19.2 Format of a Covariance Matrix File....106
5. Running PEST....108
5.1 Introduction....108
5.1.1 General....108
5.1.2 Checking PEST's Input Data....108
5.1.3 Versions of PEST....108
5.1.4 Starting PEST....109

5.1.5 Command Line Switches 109
5.2 The PEST Run Record File 114
5.2.1 An Example 114
5.2.2 Echoing the Input Data Set 123
5.2.3 The Parameter Estimation Record 123
5.2.4 Optimised Parameter Values and Confidence Intervals 125
5.2.5 Observations and Prior Information 126
5.2.6 Objective Function 127
5.2.7 Correlation Coefficient 127
5.2.8 Analysis of Residuals 127
5.2.9 Kullback-Leibler (K-L) Information Loss Statistics 127
5.2.10 Post-Calibration Parameter Covariance Matrix 128
5.2.11 Correlation Coefficient Matrix 129
5.2.12 Normalised Eigenvector Matrix and Eigenvalues 129
5.3 Other PEST Output Files 130
5.3.1 General 130
5.3.2 Parameter Value File 130
5.3.3 Parameter Sensitivity File 131
5.3.4 Observation Sensitivity File 134
5.3.5 Residuals File 135
5.3.6 Interim Residuals File 136
5.3.7 The Matrix File 136
5.3.8 The Condition Number File 137
5.3.9 Singular Value Decomposition File 138
5.3.10 LSQR Output File 138
5.3.11 Run Management Record File 138
5.3.12 Pareto Output Files 138
5.3.13 The Jacobian Matrix File 138
5.3.14 Resolution Data File 139
5.3.15 Other Files 139
5.3.16 PEST's Screen Output 139
5.3.17 Run-time Errors 140
5.4 Stopping and Restarting PEST 141
5.4.1 Interrupting PEST Execution 141
5.4.2 Restarting PEST 141
5.4.3 Why Stop PEST? 141
5.5 If PEST Won't Work 142
5.5.1 General 142
5.5.2 Model Output File not Found 142
5.5.3 Objective Function Gradient Zero 142
5.5.4 Erratic Objective Function Behaviour caused by Bad Derivatives 143
5.5.5 Other Factors Leading to Erratic Objective Function Behaviour 144
5.5.6 Excessive Number of Model Runs 144
5.5.7 Discontinuous Problems 144
5.5.8 Parameter Correlation and Insensitivity 145
5.6 PEST Postprocessing 145

5.6.1 General 145
5.6.2 Calibration as Hypothesis-Testing 145
5.6.3 Traditional Statistics 147
5.6.4 Statistics Appropriate to Regularised Inversion 147
5.6.5 Information Content of the Calibration Dataset 147
5.6.6 Model Outputs based on Optimal Parameter Values 148
6. Intervention 149
6.1 Introduction 149
6.2 User Intervention 149
6.2.1 Aberrant PEST Behaviour in the Absence of Regularisation 149
6.2.2 Fixing the Problem 150
6.2.3 The Parameter Hold File 150
6.2.4 Re-calculating the Parameter Upgrade Vector 152
6.2.5 Maximum Parameter Change 153
6.3 Automatic User Intervention 153
6.3.1 Concepts 153
6.3.2 Automatic User Intervention Control Variables 154
6.3.3 A Note on Default Automatic User Intervention Settings 158
6.4 Bad Derivatives Mitigation 159
6.4.1 General 159
6.4.2 Implementation 159
7. Sensitivity Reuse 161
7.1 Introduction 161
7.2 Implementation Details 161
7.2.1 Activating Sensitivity Reuse 161
7.2.2 Sensitivity Reuse Control Variables 161
8. Predictive Analysis 164
8.1 Introduction 164
8.2 Predictive Analysis Mode 165
8.3 Predictive Analysis Control Variables 167
8.3.1 The PESTMODE Variable 167
8.3.2 Predictive Analysis Section of the PEST Control File 167
8.3.3 User Intervention 173
8.4 Confidence and Predictive Intervals 173
8.4.1 General 173
8.4.2 One Methodology for Accommodation of Predictive Noise 174
8.4.3 An Alternative Methodology 175
8.4.4 Objective Function Constraints 175
8.4.5 Implementation in PEST 176
8.5 Model-Based Hypothesis Testing 176
9. Tikhonov Regularisation 178
9.1 General 178
9.1.1 Advantages of Highly Parameterized Inversion 178
9.1.2 Tikhonov Regularisation 178
9.1.3 Measurement and Regularization Objective Functions 179
9.1.4 Target Measurement Objective Function 180

9.1.5 Solution Mechanism....181
9.1.6 Termination Criteria....181
9.2 Regularisation and the PEST Control File....182
9.2.1 Setting the Mode....182
9.2.2 Observation Groups....182
9.2.3 Regularisation Section....182
9.3 Regularisation Weight Relativity Adjustment....187
9.3.1 Overview....187
9.3.2 IREGADJ Settings of 1 and 2....188
9.3.3 IREGADJ Setting of 3....189
9.3.4 IREGADJ Settings of 4 and 5....190
9.4 PEST Run-Time and End-of-Run Information....193
9.4.1 Run-Time Information....193
9.4.2 Post-Run Information....194
9.5 Group-Specific Target Objective Functions....195
9.5.1 General....195
9.5.2 Implementation....197
9.6 A Final Word....197
10. SVD-Assist....199
10.1 Concepts....199
10.1.1 Super Parameters....199
10.1.2 Implementation Overview....200
10.1.3 How Many Super Parameters?....202
10.1.4 Strengths and Weaknesses....202
10.2 SVDAPREP....203
10.2.1 Preparing a PEST Input Dataset....203
10.2.2 Running SVDAPREP....204
10.2.3 What SVDAPREP Does....206
10.2.4 The New Model Batch File....207
10.3 The PARCALC and PICALC Utilities....208
10.3.1 PARCALC....208
10.3.2 PICALC....208
10.4 The Super Parameter PEST Control File....209
10.4.1 General....209
10.4.2 An Important Note on Super Parameter Definition....209
10.4.3 The “SVD Assist” Section....210
10.5 SVD-Assisted Inversion....217
10.5.1 What PEST Does....217
10.5.2 Best Parameter Estimates....218
10.5.3 Parallelised SVD-Assisted Inversion....218
10.6 Some Final Points....219
10.6.1 JCO2JCO and other Utility Programs....219
10.6.2 Efficiency....219
10.6.3 Null Space Monte Carlo....220
11. Parallel PEST and BEOPEST....223
11.1 General....223

11.1.1 Introduction 223
11.1.2 Parallelisation of the Jacobian Matrix Calculation Process 223
11.1.3 Parallelisation of the Parameter Upgrade Process 224
11.1.4 Communication Overheads 224
11.1.5 Installing Parallel PEST and BEOPEST 224
11.1.6 The Parallel Run Queue 225
11.2 Parallel PEST – Concepts and Specifications 225
11.2.1 Model Input and Output Files 225
11.2.2 The Parallel PEST Slave Program 226
11.2.3 Running the Model on Different Machines 227
11.2.4 Communications between Parallel PEST and its Slaves 227
11.2.5 The Parallel PEST Run Management File 228
11.2.6 Alternative PARLAM Settings 233
11.2.7 PARLAM Override 235
11.2.8 Slave Groups 235
11.3 Using Parallel PEST 236
11.3.1 Preparing for a Parallel PEST Run 236
11.3.2 Starting the Slaves 236
11.3.3 Starting PEST 237
11.3.4 Re-Starting Parallel PEST 238
11.3.5 Starting Slaves Late 238
11.3.6 Losing Slaves 239
11.3.7 Re-Starting Slaves 239
11.3.8 Parallelisation of First Model Run 239
11.3.9 The Parallel PEST Run Management Record File 240
11.3.10 The Importance of the WAIT Variable 240
11.3.11 If Parallel PEST does not Respond 241
11.3.12 An Example 242
11.4 BEOPEST 242
11.4.1 Credits 242
11.4.2 Some Concepts 243
11.4.3 Versions of BEOPEST 244
11.4.4 Running BEOPEST as the Master 244
11.4.5 Running BEOPEST as the Slave 244
11.4.6 Terminating BEOPEST Execution 245
11.4.7 BEOPEST and SVD-Assist 245
11.4.8 Parallelisation of Initial Model Run 246
11.4.9 Multiple Command Lines 246
11.4.10 The Run Management File, PARLAM and RUN_SLOW_FAC 246
11.4.11 Run Management Record File 248
11.4.12 Slave Groups 248
11.4.13 Culling Slaves 248
11.4.14 Opening a Port to the Outside World 249
11.4.15 An Example 249
12. Model-Calculated Derivatives 250
12.1 Contents of this Chapter 250

12.2 Sending a Message to the Model 250
12.3 Multiple Command Lines 251
12.4 Externally-Supplied Derivatives 252
12.4.1 General 252
12.4.2 External Derivatives File 252
12.4.3 File Formats 253
12.4.4 File Management 254
12.4.5 Derivatives Type 254
12.4.6 Tied Parameters 255
12.4.7 Combining External and Finite-Difference Derivatives 255
12.4.8 Name of the Derivatives File 256
12.4.9 Predictive Analysis Mode 256
12.4.10 Parallel PEST and BEOPEST 256
12.5 PEST Control Variables 256
12.5.1 General 256
12.5.2 “Control Data” Section 256
12.5.3 “Parameter Data” Section 257
12.5.4 “Derivatives Command Line” Section 258
12.5.5 “Model Command Line” Section 258
12.6 JUPITER Protocol for External Derivatives 259
12.6.1 General 259
12.6.2 JUPITER Protocol for Model-Calculated Derivatives 259
12.6.3 Using JUPITER Derivatives Protocol with PEST 261
12.6.4 Special Considerations 261
12.7 Model-Calculated Derivatives and SVD-Assist 262
12.7.1 External Derivatives 262
12.7.2 SVDAPREP 263
12.8 An Example 264
13. Pareto Mode 267
13.1 The Pareto Front 267
13.2 Exploration of Predictive Credibility 268
13.2.1 Concepts 268
13.2.2 Implementation 269
13.2.3 Preparations for a PEST Pareto Run 271
13.2.4 PEST Output Files 272
13.3 Highly Parameterized Inversion as a Trade-off 274
13.3.1 Concepts 274
13.3.2 Implementation 274
13.3.3 Operational Details 276
13.4 Some Further Implementation Details 277
13.4.1 Parallelisation and Pareto 277
13.4.2 Restarting a Pareto Run. 277
13.4.3 Special Considerations for SVD-Assist 277
13.4.4 Going the Other Way 279
13.4.5 Some Differences and Similarities with Other PEST Modes 279
13.5 Pareto Support Utilities 280

13.5.1 The PPD2ASC Utility 280
13.5.2 The PPD2PAR Utility 281
14. Observation Re-referencing 282
14.1 Introduction 282
14.2 General Principles 282
14.3 Two Observation Re-Referencing Modes 283
14.3.1 Mode 1 283
14.3.2 Mode 2 284
14.4 Activating Observation Re-Referencing 285
14.5 SVD-Assist and Observation Re-referencing 285
14.6 Parallelisation 286
14.7 Restarting 286
15. Large Numbers of Parameters 288
15.1 Introduction 288
15.2 Derivatives Calculation 288
15.3 Versions of PEST 288
15.4 Compressed Internal Jacobian Storage 288
15.4.1 Concepts 288
15.4.2 The MAXCOMPDIM Control Variable 289
15.4.3 The DERZEROLIM Control Variable 290
15.5 Ordering of Observation Groups 290
15.6 Linear Regularisation Constraints 290
15.7 Trading Memory for Functionality 291
15.8 Accelerated Input of Prior Information 292
15.9 Accelerating Input of External Derivatives 294
15.10 Other Savings 295
16. PEST-Compatible Global Optimisers 296
16.1 CMAES_P 296
16.1.1 General 296
16.1.2 The Algorithm in Brief 296
16.1.3 The CMAES_P Implementation of CMA-ES 297
16.1.4 Preparing for a CMAES_P Run 298
16.1.5 Running CMAES_P 300
16.1.6 Supplying a Parameter Covariance Matrix 304
16.1.7 SVD-Hybridization 305
16.1.8 CMAES_P Output Files 307
16.1.9 Parallelisation 308
16.1.10 Pseudo-Regularisation 308
16.2 SCEUA_P 310
16.2.1 Introduction 310
16.2.2 Using SCEUA_P 310
16.2.3 Parallel SCEUA_P 315
16.2.4 Problem Formulation 317
17. SENSAN 318
17.1 Introduction 318
17.2 SENSAN File Requirements 319

17.2.1 General 319
17.2.2 Template Files 319
17.2.3 Instruction Files 319
17.2.4 The Parameter Variation File 320
17.2.5 SENSAN Control File 321
17.2.6 Control Data Section 322
17.2.7 SENSAN Files Section 323
17.2.8 Model Command Line Section 323
17.2.9 Model Input/Output Section 323
17.2.10 Issuing a System Command from within SENSAN 324
17.3 Running SENSAN 324
17.3.1 SENSAN Command Line 324
17.3.2 Interrupting SENSAN Execution 325
17.4 Files Written by SENSAN 325
17.4.1 SENSAN Output Files 325
17.4.2 Other Files used by SENSAN 326
17.5 Sensitivity of the Objective Function 326
17.6 SENSAN Error Checking and Run-Time Problems 327
17.7 An Example 328
18. A Simple PEST Example 330
18.1 Parameter Estimation 330
18.1.1 Laboratory Data 330
18.1.2 The Model 331
18.1.3 Preparing the Template File 334
18.1.4 Preparing the Instruction File 335
18.1.5 Preparing the PEST Control File 336
18.2 Predictive Analysis 338
18.2.1 Obtaining the Model Prediction of Maximum Likelihood 338
18.2.2 The Composite Model 339
18.2.3 The PEST Control File 340
18.2.4 Template and Instruction Files 341
18.2.5 Running PEST 342
19. References 344
Appendix A. PEST Control File Specifications 346
Appendix B. Files Used by PEST 363
B1.1 General 363
B1.2 PEST Input Files 363
B1.3 PEST Output Files 364

1. Introduction

1.1 Installation

Installation of PEST is simple. Copy all executable programs that are supplied with PEST to a suitable folder. Then make sure that this folder is cited in the PATH environment variable so that your machine can find PEST executable programs regardless of your current working directory.

1.2 Software

The "PEST suite" comprises

- PEST itself;
- variants of PEST which facilitate parallel run management (namely Parallel PEST and BEOPEST);
- two so-called "global optimisers" that can be used as direct replacements for PEST;
- a basic sensitivity analyser named SENSAN;
- a suite of utility support programs that are supplied with PEST itself; and
- other utility support programs that expedite use of PEST in specific modelling contexts such as groundwater and surface water modelling.

PEST, Parallel PEST, BEOPEST, SENSAN and the global optimisers are described in the present document, which comprises part I of the PEST manual. Part II of the manual describes the utilities which support the use of PEST in the following and other ways:

- facilitation of construction of PEST input datasets;
- checking of the integrity of PEST input datasets;
- processing and summarising the information contained in PEST output datasets;
- processing and manipulation of sensitivity matrices;
- general matrix manipulation;
- linear parameter and predictive identifiability and uncertainty analysis;
- linear data worth analysis;
- analysis of calibration-induced parameter and predictive bias arising from model defects;
- facilitation of nonlinear parameter and predictive uncertainty analysis;
- pre- and post-calibration random parameter generation.

Broader utility software which expedites the use of PEST in specific modelling contexts, and which expedites PEST setup for use with models that are commonly used in these contexts, have their own documentation. At present two such broad suites of utility software are available, namely the PEST Groundwater Utility suite and the PEST Surface Water Utility suite. Note, however, that the use of programs within these suites is not restricted to just groundwater modelling on the one hand and surface water modelling on the other hand. Indeed, members of the Groundwater Utility suite, and the parameter list processor PLPROC, have been used successfully with petroleum, gas, and geothermal subsurface reservoir models. Similarly, the surface water suite has been used extensively with land use and recharge models. These PEST-support utility suites, and their associated documentation, can be downloaded, along with PEST itself, from the PEST web pages at

http://www.pesthomepage.org

1.3 Concepts

Historically, the first program that comprised the PEST suite was PEST itself. PEST stands for "parameter estimation". It was originally written to expedite the process of model calibration wherein values for model parameters are back-calculated by matching model outputs to measurements of system state. "Parameters" employed by a model can represent the properties of the materials in which processes simulated by the model take place, the stresses which initiate and support those processes, or both. What made PEST different from parameter estimation software which preceded it was the fact that PEST operates in a model-independent manner; it interacts with a model through the model's own input and output files. Hence no programming is required to use PEST to calibrate a model.

Model calibration falls into a broad class of numerical methods which mathematicians describe as "inversion". In general, solution of an inverse problem is nonunique. However it can be re-caste as unique if only a small number of parameters is estimated. The process of parameter simplification that is a necessary precondition for attainment of uniqueness is known as "regularisation". This can be done manually prior to estimating parameters. Or it can be done by the inversion process itself. In general it is better to do it the latter way. PEST accommodates both.

Because of the generally nonunique status of the inverse problem whose solution is required for calibration of a model, and because the observation dataset which is used for model calibration is generally noisy, parameters estimated through the model calibration process are uncertain. Furthermore the relationships between parameters estimated through inversion and the actual system properties which they represent can be unclear. One of the benefits of adopting a highly parameterized approach to inversion, in which the regularisation required for attainment of inverse problem solution uniqueness is accomplished as part of the solution process itself, is that the issues of parameter uncertainty, and of the relationships between estimated parameters and their real-world counterparts, can then be explored. In fact, where the relationship between model parameters and model outputs can be approximated as linear, exploration of these issues becomes easy. The PEST suite includes a number of programs which are designed for this purpose. Some of these programs have their roots in Bayes equation; others employ subspace methods based on singular value decomposition. All of them provide interesting insights into the inversion process.

The PEST suite also provides methods for exploration of pre- and post-calibration parameter uncertainty which do not require an assumption of model linearity. PEST input files can be populated with random parameter sets; model runs can then be undertaken using each of these sets so that the variability of model outputs of interest can be explored. Random parameter sets can be constrained, using the so-called "null space Monte Carlo" method which is unique to PEST, so that model outputs fit calibration datasets; the null space Monte Carlo methodology uses subspace concepts to accomplish the normally difficult task of enforcing parameter constraints on random parameter values in a numerically efficient manner.

If a model's parameters are uncertain, then so too are its predictions. Quantification of model predictive uncertainty is essential to model-supported environmental decision-making; if predictive uncertainty cannot be assessed, then it is not possible to include the vital element of risk in the decision-making process. The PEST suite provides both linear and nonlinear methodologies for quantification of predictive uncertainty. Some are extensions of the

parameter uncertainty assessment methods mentioned above. Others are more "prediction focussed". The latter include PEST's predictive analysis functionality and PEST's Pareto capabilities. Both of these can be used to directly test the hypothesis that the value of a particular model prediction is compatible with both expert knowledge (as encapsulated in reasonableness of parameter values) and the historical behaviour of a system (as encapsulated in a calibration dataset).

Further functionality provided through the PEST suite allows a modeller to explore the effects of model defects on predictions made by a model, both before and after the model's parameters have been subjected to history-matching constraints. All environmental models are defective because all environmental models are simplifications of reality. This has important repercussion for the way in which a model should be calibrated, and for assessing the credibility of the different types of predictions which a model is asked to make.

1.4 Philosophy

At the time of writing, the PEST suite is by far the most advanced software available for environmental model calibration, and for post-calibration uncertainty analysis. As such, it holds a unique place in the environmental industry – not just for what it can do, but for what it can teach us about environmental modelling.

PEST, and its ancillary software, embraces the fact that environmental systems are complex, and that a model's parameters and predictions are uncertain. Its use supports the critical notion that a model can never tell us what will happen in the future following adoption of a certain environmental management practice; this is an outcome of the uncertainties associated with most model predictions of environmental interest. However a model may tell us, with a high degree of confidence, what will NOT happen in the future. In order to accomplish this, it cannot be deployed on its own; instead it must be deployed in conjunction with high-end inversion software such as PEST. Under these circumstances models may then provide invaluable support to the decision-making process by allowing rejection of hypotheses that unwanted events will occur if certain courses of management action are taken. (These courses of management action will often include installation of monitoring systems which trigger responses to the crossing of pre-defined measurement thresholds.)

It follows that a model, as a simulator, does not constitute a decision-support tool. In contrast, the model, as a simulator, should constitute one of a number of software packages, which are used in partnership to

- quantify the uncertainties associated with predictions of management interest, and
- reduce those uncertainties to a level that is commensurate with information available from expert knowledge on the one hand and the historical behaviour of a system on the other hand.

Software packages used in concert to achieve these ends collectively constitute an indispensable tool for decision-support – a tool which embraces the complexity of the real world at the same time as it provides decision-makers with an understanding of the ramifications of that complexity for the decisions that they must make.

For a full discussion of the role of models in decision-making see Doherty and Simmons (2013) and Doherty and Vogwill (2016).

1.5 PEST- The Book

The following book can be downloaded from the PEST web site:

Doherty, J., 2015. Calibration and uncertainty analysis for complex environmental models. Published by Watermark Numerical Computing, Brisbane, Australia. 227pp, ISBN: 978-0-9943786-0-6

The numerous references to Doherty (2015) throughout this manual are to this text.

Doherty (2015) covers in details all of the theory embodied in PEST and its utility support software. It also discusses the ramifications of this theory for how models should be used in real-world environmental decision-making. It provides an extensive discussion on the theory and practice of regularisation – whether this is done manually through parameter simplification, or mathematically using subspace or Tikhonov methods. It also provides a critique of manual regularisation, and provides an in-depth discussion on why it is important to reflect environmental system complexity in model parameterisation complexity if models are to play a useful role in support of environmental decision-making. It shows how parameter nonuniqueness is not something to run away from, but something to embrace; after all, it is the parameters that *cannot* be estimated that contribute most to predictive uncertainty - generally much more than those which *can* be estimated. It demonstrates that the use of many parameters in an inversion problem does not necessarily lead to over-fitting; nor does it promulgate numerical instability, or result in solutions to the inverse problem which are unnecessarily complex. The book shows that parameter simplification is fundamental to achievement of a unique solution to an inverse problem, and that achievement of parameter simplification through mathematical means leads to reduction of potential for model predictive error at the same time as it allows quantification of this potential for error.

Parts I and II of this manual refer to the above book extensively. They do not repeat the theory presented in the book; nor do they discuss the ramifications of that theory for model deployment in the decision-making context. This enables these manuals to be somewhat shorter than they would otherwise be; it also enables them to concentrate on implementation details. As a PEST user, you are strongly advised to read the book, for this will provide you with the theoretical basis that you need to get the most out of PEST.

1.6 Some Practical Considerations

1.6.1 The Model

In the course of estimating its parameters, and/or of computing sensitivities of model outputs to parameters, PEST must run a model many times. It does this through a call to the operating system (known as a "system call"). This call has the same effect as typing the name of the model (and any arguments required by the model which follow its name) in a command line window (sometimes called a "DOS box" on a WINDOWS system). Hence the model must be accessible to a user (and therefore to PEST) through the command line. Ideally, the directory (i.e. folder) in which the model executable resides should be featured in the PATH environment variable so that the operating system knows where to find it.

Some models are not accessible to the user in this way. Some models can only be run through their own model-specific graphical user interface; they may be called as a DLL by this interface. Models which can only be accessed in this way cannot be used with PEST.

Nevertheless, there is a considerable amount of flexibility in what constitutes "a model" that

can be run by PEST. A model can be a batch (on a WINDOWS system) or script (on a UNIX system) file which runs one or a number of executable programs in succession. Hence a simulator can be accompanied by preprocessors and postprocessors; the former may manipulate parameters (for example interpolate pilot point parameters to the cells which constitute a model grid), while the latter may manipulate model outputs (for example interpolate model outputs from grid cell centres to the locations of observation points). The model may actually be comprised of a number of simulators which are run in succession (for example a flow model followed by a transport model). Where this is the case, it is a good idea to commence such a batch file with the command to delete all files which are written by one executable program and then read by another. Thus if the first executable does not run, the second executable will not read an old file generated by the first executable, mistaking it for a new one. It will therefore crash. The final output files which PEST must read will then be absent (PEST deletes these files before it runs the model). PEST will then cease execution gracefully with an appropriate error message.

Sometimes, in the WINDOWS environment, a model whose execution is initiated from the command line, or through a system call, returns control to the calling program (or the user prompt) immediately, even though it is still running. This would cause PEST to look for its output files, thinking that it has finished. This can be prevented by using the "start /w" command. Type

```
start /?
```

at the command line prompt for more details pertaining to the WINDOWS "start" command.

If, on commencement of execution, a model prompts the user for keyboard input, this situation can be easily accommodated. The keyboard responses to the model's prompts can be placed into a text file. Suppose that this file is named *model.inp*. Suppose also that the name of the model executable file is *model.exe,* or simply *model* on a UNIX platform. Then if the model is run using the command

```
model < model.inp
```

it will look to file *model.inp* rather than the keyboard for its input. PEST can run the model using this command without the need for any user involvement.

1.6.2 Model Input and Output Files

As is described herein, PEST can interact with a model through the model's input and output files. A model can have many input files and many output files; PEST can interact with all of these; the number of input and output files does not matter.

So-called "template files" must be written, based on model input files, to allow PEST to recognize those parts of a model input file which it must change before running the model. Alterations to a model's input file are only required for the purpose of providing the model with a set of parameter values which are appropriate for that model run. A thus-altered model input file must be an ASCII file; it cannot be a binary file. Hence even if a particular model reads much of its input dataset from one or more binary files, the file or files which contain parameter values must be ASCII (i.e. text) files.

Similar considerations apply to model output files. PEST reads those numbers from model output files for which there are corresponding field measurements using instructions contained in so-called "instruction files". The files from which these numbers are read must be ASCII files. If the model writes its outputs to a binary file, then a postprocessor which follows the model in a model batch or script file must be used to rewrite pertinent model

outputs in ASCII format.

1.6.3 WINDOWS and UNIX

Sometimes difficulties can be experienced by either PEST or a model if a template file is prepared on a UNIX system and then used on a WINDOWS system. The latter operating system terminates each line of a text file with the carriage return and line feed characters; UNIX only uses the line feed. Readily downloadable utility programs such as DOS2UNIX and UNIX2DOS allow conversion between the two protocols.

1.6.4 Differentiability

Methods used by PEST for inversion and calibration-constrained uncertainty analysis fall under the general term of "gradient methods". These are powerful methods, as the number of parameters that they can accommodate can be very large indeed – even in the hundreds of thousands. However their use requires that derivatives of model outputs used in the inversion process with respect to parameters adjusted through that process be calculable. PEST calculates these derivatives using finite differences; it varies a parameter incrementally, and then divides the incremental change in pertinent model outputs by the incremental change in the parameter.

If numbers calculated by a model are not continuous functions of the model's parameters this process will fail. Non-continuity of model outputs with respect to parameters can be an outcome of poor model design, or it can be an inadvertent outcome of numerical difficulties that a model may experience – particularly if it employs an iterative scheme to solve the equations which describe the processes which it simulates.

PEST can accommodate problematical continuity of model outputs with respect to parameters in a number of ways. It can base computation of finite-difference derivatives on the use of three, or even five, parameter values instead of just two; this, of course, comes at a cost in terms of model runs. It can also detect and reject problematical derivatives. However if a model's performance is too bad in this respect, for example if a model uses categorical parameters which must adopt discrete values instead of continuously variable values, then use of PEST with that model becomes impossible. In this case a modeller must employ a so-called "global method" which does not base its estimation of a calibrated parameter set on continuity of model outputs with respect to adjustable parameters. Two global optimisers are provided with the PEST suite, names SCEUA_P and CMAES_P.

Use of global methods is accompanied by some disadvantages however. These include the following.

- global methods cannot handle the large number of parameters that gradient methods can;
- they can struggle where an inverse problem is ill-posed;
- they are usually far less model-run-efficient in finding a solution to an inverse problem than are gradient methods;
- it is difficult, if not impossible, to use advanced regularisation methods with global optimisers;
- linear parameter uncertainty and identifiability statistics that are rapidly computed from derivatives of model outputs with respect to adjustable parameters are not available.

Often simple steps can (and should) be taken to make use of a model more tractable with PEST. As stated above, PEST reads model outputs from files written by the model. These

outputs should be recorded with as much numerical precision as the model allows (seven significant figures if the model uses single precision arithmetic). When PEST computes derivatives using finite differences, it must subtract numbers which are very similar. Leading significant figures are lost in this process, leaving only the trailing significant figures. If these are not represented, then finite-difference derivatives lose their integrity.

Similarly, the convergence criteria of iterative solvers employed by a model should be set tightly. The loss of a few significant figures incurred by loose convergence criteria matters little as far as model predictions are concerned. However they inflict a heavy cost on finite-difference derivatives. Sometimes it may also be necessary to curtail the activities of numerical devices such as adaptive time stepping if it is felt that they may unwittingly contribute alterations to model outputs that do not arise exclusively from alterations to the values of model parameters.

1.6.5 Local Optima

Extreme inverse problem nonlinearity may lead to local optima in the calibration objective function. A common complaint made against gradient methods is that they may find a local optimum instead of the global optimum.

This assertion must be seen in context. While the existence of local optima in some calibration contexts is undeniable, there have been many occasions in the literature where parameter nonuniqueness born of the existence of a null space, has been mistaken for parameter nonuniqueness born of local optima. Gradient methods, with their ability to incorporate sophisticated regularisation schemes in the inversion process, work comfortably in the former context. Regularisation can help in the latter context as well, as can formulation of a multi-component objective function that reflects the information content of various components of the observation dataset; see White et al (2014) and Doherty (2015) for details. Furthermore, as Kavetski et al (2006) point out, sometimes local optima arise as an unwanted outcome of a poor model algorithm.

1.6.6 Observations and Predictions

As stated above, PEST is able to read numbers of interest from model output files using instructions that are recorded in instruction files. For early versions of PEST such numbers are those for which there are measured counterparts. Collectively these numbers comprise the model-generated counterpart to the calibration dataset. For more recent versions of PEST some of these numbers may be predictions for which there are no measured counterparts.

Predictions may feature in a PEST dataset for a number of reasons. When PEST is run in "predictive analysis" mode it is instructed to maximise or minimise a prediction subject to maintaining calibration constraints, i.e. subject to maintaining the calibration objective function below a certain threshold. When PEST is run in "pareto" mode, the weight on one or a number of predictions may be slowly increased as "fitting the prediction" is traded off against fitting the calibration dataset.

A modeller may include predictions in a PEST dataset so that sensitivities of the prediction to parameters employed by the model can be calculated. PEST may be run solely to calculate these sensitivities. Alternatively they may be "carried" in an inversion process for the same reason while PEST is actually engaged in adjusting parameters so that model outputs better match field data. In the latter of these cases, PEST does not require that pertinent model outputs be specifically identified as predictions. By providing them with weights of zero their effect on the parameter estimation process is neutralized.

Any number read from a model output file is referred to as "an observation" in PEST parlance. If it is given a weight of zero, it is therefore not matched to a corresponding measurement (though PEST input protocols still require that the value of a measurement be provided). Hence some "observations" can in fact be predictions. As such, they may feature in linear and nonlinear predictive uncertainty analysis.

1.6.7 Numerical Burden

Parameter estimation and calibration-constrained parameter and predictive uncertainty analysis requires the undertaking of many model runs. Where model run times are long, and where the number of parameters requiring adjustment is large, this represents a considerable numerical burden. PEST provides three mechanisms for easing this burden.

The first mechanism is run parallelisation. Calculation of a Jacobian (i.e. sensitivity) matrix is 100 percent parallelizable. To some extent the testing of parameter upgrades calculated on the basis of these sensitivities using different values of the Marquardt lambda is also parallelizable. Parallel PEST and BEOPEST allow a modeller to parallelise runs across different processors on the same machine, across machines, across networks and across the world.

When undertaking highly parameterized inversion, the numerical burden of Jacobian matrix calculation can be greatly eased if sensitivities are calculated not to parameters themselves, but to *combinations* of parameters that have been identified through singular value composition as being uniquely estimable on the basis of the current calibration dataset; these combinations are referred to as "super parameters" in PEST parlance. PEST accomplishes this through its SVD-assist functionality. Model runs undertaken for super parameter sensitivity calculation can be parallelised, this achieving further increases in numerical efficiency.

Through its "observation re-referencing" functionality, PEST can use one or more surrogate models instead of the actual model when calculating finite-difference derivatives of model outputs with respect to adjustable parameters. If the run time of the surrogate model is much smaller than that of the actual model, the computational burden of the inversion process can be reduced considerably. Further gains can be made through parallelisation of surrogate and actual model runs in an SVD-assisted inversion process.

1.7 Some Common Tasks

1.7.1 General

The purpose of this section is to identify a number of tasks that can be undertaken using the PEST family of software, and to list the PEST-suite programs which can be used to carry out these tasks. Even though reference is made to files and concepts which are explained later in this manual, this section introduces a new user to the capabilities offered by PEST and its ancillary utility software.

1.7.2 Over-Determined Model Calibration

Where a small number of parameters must be estimated, there is no need to include regularisation in a PEST control file. The PESTGEN utility can assist in PEST input file preparation. However programs provided with the PEST Groundwater and Surface Water Utility suite may provide more comprehensive assistance in PEST input dataset construction. The integrity of PEST template and instruction files can be checked using the TEMPCHEK

and INSCHEK utilities. The integrity of the entire PEST input dataset can be checked using the PESTCHEK utility.

After one model run has been completed, weights balancing between observation groups featured in the PEST dataset can be implemented using PWTADJ1. WTFACTOR provides further assistance in weights adjustment were observation numbers are large.

Upon completion of the parameter estimation process, EIGPROC can summarize information that is pertinent to the well-posedness (or otherwise) of that process. PHISTATS can provide a summary of objective function behaviour during the inversion process. If you wish to build a new PEST dataset starting with parameters calculated during the current PEST run, you can use the PARREP utility to populate the new PEST control file accordingly.

1.7.3 Highly Parameterized Inversion

Once a PEST input dataset has been prepared, the ADDREG1 utility can be used to rapidly add "preferred value" Tikhonov regularisation to a PEST control file. PEST can then be run without the need for any further input file preparation. On completion of the PEST run (or only one iteration of a PEST run in which a Jacobian matrix is calculated), the SSSTAT utility can be used to obtain a complete range of parameter and observation statistics based on subspace concepts. IDENTPAR can be used to compute parameter identifiabilities; alternatively GENLINPRED can be used to compute both parameter identifiabilities and relative parameter uncertainty reductions. A linear approximation to the posterior parameter covariance matrix can be obtained using PREDUNC7.

If inversion is to be carried out using the efficient SVD-assist methodology, the SUPCALC utility can be employed to infer the dimensionality of the calibration solution space, and hence place a lower bound on the number of super parameters that must be estimated. SVDAPREP can then be run to automate PEST input dataset preparation for SVD-assisted inversion.

1.7.4 Parameter Preprocessing and Observation Postprocessing

Creativity in definition of inversion-relevant parameters can be enhanced by running the PAR2PAR parameter preprocessor prior to running the simulator in the batch or script file which PEST runs as "the model". Optimality in definition of observations, and in formulation of a multi-component objective function that is tuned to the information content of a calibration dataset, and that provides some defence against the deleterious effects of structural noise, can be achieved by running the OBS2OBS utility after the simulator in the model batch or script file.

1.7.5 Linear Sensitivity and Uncertainty Analysis

Upon completion of a PEST run the SUBREG1 utility can be used to remove regularisation from a PEST control file. Prior to post-calibration linear analysis, the PWTADJ2 utility can adjust weights so that they reflect measurement and structural noise as established during the preceding inversion process. PREDUNC1, PREDUNC4, PREDUNC5 and PREDUNC6 can be used to explore parameter and predictive uncertainty, contributions to uncertainty made by different parameters, and the worth (or otherwise) of different data types in reducing predictive uncertainty. The PREDVAR1, PREDVAR4 and PREDVAR5 utilities accomplish the same roles, but explore parameter and predictive error variance rather than parameter and predictive uncertainty (for the difference between "error" and "uncertainty" see Doherty, 2015). Alternatively, all of these utilities can be run using the GENLINPRED user interface.

Further mechanisms for exploring observation worth are provided through the INFSTAT and INFSTAT1 utilities. Meanwhile the SUPOSBPAR and SUPOBSPAR1 utilities can be used to establish combinations of observations comprising the calibration dataset which are uniquely and entirely informative of corresponding combinations of parameters.

The propensity for calibration-induced parameter and predictive error arising from model defects can be explored using the PREDVAR1B and PREDVAR1C utilities.

1.7.6 Jacobian and General Matrix Manipulation

The filling of a Jacobian matrix may require that many model runs be undertaken. A number of utilities are provided that manipulate, transform, decompose and re-compose this matrix. Among other benefits, these can reduce the need to waste model runs in re-computing this matrix if the parameter and/or observation composition of the inverse problem is altered.

JCO2JCO and JCOCOMB can be used to build a Jacobian matrix file (i.e. a "JCO file") for a new PEST control file which features the same, or fewer, observations and/or parameters as an existing PEST control file. JCOPCAT, JCOORDER, JCOSUB and JCOADDZ assist in formulating a new Jacobian matrix from partial Jacobian matrices.

JROW2VEC can be used to extract a single row from a Jacobian matrix. This can be useful in linear predictive uncertainty analysis where that row contains sensitivities of a prediction to adjustable parameters.

JACWRIT rewrites a binary Jacobian matrix file as an ASCII file. JCOCHEK checks the compatibility of a Jacobian matrix file with a PEST control file. JCO2MAT rewrites a Jacobian matrix contained in a JCO file in PEST matrix file format where it can be subjected to matrix-based analysis and/or transferred between WINDOWS and UNIX platforms. MAT2JCO does the opposite to this. WTSENOUT replaces a Jacobian matrix with a weighted Jacobian matrix.

Utility software documented in part II of this manual also includes a collection of programs which perform a suite of inversion-relevant matrix operations that include singular value decomposition, and formulation of a so-called "normal matrix" from a Jacobian matrix.

1.7.7 Testing the Integrity of Finite-Difference Derivatives

The JACTEST utility can run a model a user-specified number of times using parameter increments that are the same as those that PEST uses when calculating finite-difference derivatives. The POSTJACTEST utility assists in processing and plotting the outcomes of these runs in search of signs of model numerical malperformance.

If JACTEST and POSTJACTEST establish that finite-difference derivatives do not have adequate numerical integrity to support the use of PEST, then the SCEUA_P and CMAES_P global optimisers can be used instead of PEST.

1.7.8 Nonlinear Uncertainty Analysis

RANDPAR generates random parameter sets sampled from a prior or posterior parameter covariance matrix. A linear approximation to the latter can be constructed using the PREDUNC7 utility; meanwhile COVCOND conditions a covariance matrix based on knowledge gained of some of the random variables which it describes.

Processing of the outcomes of multiple model runs undertaken to implement (post-calibration) Monte Carlo analysis can be facilitated using the COMFILNME and RDMULRES utilities. The runs themselves can be undertaken using a batch loop in which

the PARREP utility is used repeatedly to place random parameters into a PEST control file.

The PNULPAR utility is an essential component of PEST's null space Monte Carlo functionality. It uses singular value decomposition to remove the projection of a random parameter field onto the calibration solution space; this is the component of the parameter field which compromises its ability to calibrate a model. PNULPAR replaces this component with the solution space projection of the calibrated parameter set (which should, in theory, be the calibrated parameter set itself).

1.7.9 In Conclusion

The above list of tasks that can be accomplished using PEST and its associated utility suite is far from complete. Furthermore, this list of tasks would be expanded greatly if those that can be accomplished using the PEST Groundwater and Surface Water Utilities are taken into account.

See part II of this manual for documentation of utility programs supplied with PEST. Refer to the manuals of the Groundwater and Surface Water Utilities, and of the PLPROC parameter list processor, for a complete description of roles that these inversion-support utilities play in model calibration, and for complete usage details.

2. The Model-PEST Interface

2.1 PEST Input Files

PEST requires three types of input file. These are

- template files, one for each model input file in which parameters are defined;
- instruction files, one for each model output file on which model-generated observations exist, and
- a control file, supplying PEST with the names of all template and instruction files, the names of the corresponding model input and output files, the problem size, control variables, initial parameter values, measurement values and weights, etc.

This chapter describes the first two of these file types in detail; the PEST control file is discussed later in this manual Template files and instruction files can be written using a general-purpose text editor following the specifications set out in this chapter. Alternatively, they can be written using special-purpose software that is specific to a particular modelling application. Once built, they can be checked for correctness and consistency using the utility programs TEMPCHEK, INSCHEK and PESTCHEK documented in part II of this manual.

Note that in this and other chapters of this manual, the word "observation" is used to denote a number that is read from a model output file. A field measurement may correspond to this number, in which case the number is the model-generated counterpart of this measurement. If this is the case, it is the task of PEST to reduce the difference between these two (i.e. the residual) at the same time as it reduces the differences between other observations and their model-generated counterparts. In other cases the number extracted from a model output file may be given a weight of zero. In this case it is simply a model output of interest, possibly a model prediction. Nevertheless, for convenience, it is still referred to as "an observation" in this manual.

2.2 Template Files

2.2.1 Model Input Files

Whenever PEST runs a model, as it must do many times in filling a Jacobian matrix or solving an inverse problem, it must first write parameter values to the model input files which hold them. Whether the model is being run to calculate the objective function arising from user-supplied initial parameter values, to test a parameter upgrade, or to calculate the derivatives of observations with respect to a particular parameter, PEST provides a set of parameter values which it wants the model to use for that particular run. The only way that the model can access these values is to read them from its input file(s).

Some models read some or all of their data from the terminal, the user being required to supply these data items in response to model prompts. This can also be done through a file. If you write to a file the responses which you would normally supply to a model through the terminal, you can "redirect" these responses to the model using the "<" symbol on the model command line. Thus if your model is run using the command "model", and you type your responses in advance to the file *file.inp,* then you (and PEST) can run the model without having to supply terminal input using the command

```
model < file.inp
```

If *file.inp* contains parameters which PEST must optimise, a template can be built for it as if it were any other model input file.

A model may read many input files; however a template is needed only for those input files which contain parameters requiring estimation. PEST does not need to know about any of the other model input files.

PEST can only write parameters to ASCII (i.e. text) input files. If a model requires a binary input file, you must write a program which translates data written to an ASCII file to binary form expected by the model. The translator program, and then the model, can be run in sequence by listing them in a batch file which PEST runs as "the model". The ASCII input file to the translator program will then become a model input file, for which a template is required.

A model input file can be of any length. However PEST insists that it be no more than 2000 characters in width. The same applies to template files. It is suggested that template files be provided with the extension "*.tpl*" in order to distinguish them from other types of file.

2.2.2 An Example

A template file receives its name from the fact that it is simply a replica of a model input file except that the space occupied by each parameter in the latter file is replaced by a sequence of characters which identify the space as belonging to that parameter.

Consider the model input file shown in figure 2.1; this file supplies data to a program which computes the "apparent resistivity" on the surface of a layered half-space for different surface electrode configurations. Suppose that we wish to use this program (i.e. model) to estimate the properties of each of three half-space layers from apparent resistivity data collected on the surface of the half-space. The parameters for which we want estimates are the resistivity and thickness of the upper two layers and the resistivity of the third (its thickness is infinite). A suitable template file appears in figure 2.2.

```
MODEL INPUT FILE
3, 19                          no. of layers, no. of spacings
1.0, 1.0                       resistivity, thickness: layer 1
40.0, 20.0                     resistivity, thickness: layer 2
5.0                            resistivity: layer 3
1.0                            electrode spacings
1.47
2.15
3.16
4.64
6.81
10.0
14.9
21.5
31.6
46.4
68.1
100
149
215
316
464
681
1000
```

Figure 2.1 A model input file.

```
ptf #
MODEL INPUT FILE
3, 19                                  no. of layers, no. of spacings
#res1     #,#t1        #               resistivity, thickness: layer 1
#res2     #,#t2        #               resistivity, thickness: layer 2
#res3     #                            resistivity: layer 3
1.0                                    electrode spacings
1.47
2.15
3.16
4.64
6.81
10.0
14.9
21.5
31.6
46.4
68.1
100
149
215
316
464
681
1000
```

Figure 2.2 A template file.

2.2.3 The Parameter Delimiter

As figure 2.2 shows, the first line of a template file must contain the letters “ptf” followed by a space, followed by a single character (“ptf” stands for “PEST template file”). The character following the space is the “parameter delimiter”. In a template file, a “parameter space” is

identified as the set of characters between and including a pair of parameter delimiters. When PEST writes a model input file based on a template file, it replaces all characters between and including these parameter delimiters by a number representing the current value of the parameter that owns the space; that parameter is identified by name within the parameter space, between the parameter delimiters.

You must choose the parameter delimiter yourself; however your choice is restricted in that the characters [a-z], [A-Z] and [0-9] are invalid. *The parameter delimiter character must appear nowhere within the template file except in its capacity as a parameter delimiter*, for whenever PEST encounters that character in a template file it assumes that it is defining a parameter space.

2.2.4 Parameter Names

All parameters are referenced by name. Parameter references are required both in template files (where the locations of parameters on model input files are identified), in the PEST control file (where parameter initial values, lower and upper bounds and other information are provided), and in the input files of utility programs featured in the PEST suite which undertake various parameter processing tasks. Parameter names can be from one to twelve characters in length, any characters being legal except for the space character and the parameter delimiter character. Parameter names are case-insensitive.

Each parameter space is defined by two parameter delimiters; the name of the parameter to which the space belongs must be written between the two delimiters.

If a model input file is such that the space available for writing a certain parameter is limited, the parameter name may need to be considerably less than twelve characters long in order that both the name and the left and right delimiters can be written within the limited space available. The minimum allowable parameter space width is thus three characters, one character for each of the left and right delimiters and one for the parameter name.

2.2.5 Setting the Parameter Space Width

In general, the wider is a parameter space (up to a certain limit - see below), the better PEST likes it, for numbers can be represented with greater precision in wider spaces than they can be in narrower spaces. However, unlike the case of model-generated observations where maximum precision is crucial to obtaining useable derivatives, PEST can adjust to limited precision in the representation of parameters on model input files, as long as enough precision is employed such that a parameter value can be distinguished from the value of that same parameter incremented for derivatives calculation. Hence, beyond a certain number of characters, the exact number depending on the parameter value and the size and type of parameter increment employed, extra precision is not critical. Nevertheless, it is good practice to endow parameter values with as much precision as the model is capable of reading them with, so that they can be provided to the model with the same precision with which PEST calculates them.

Generally models read numbers from the terminal or from an input file in either of two ways, namely from specified fields, or as a sequence of numbers, each of which may be of any length; the latter method is often referred to as “free field” input or as “list-directed” input. If the model uses the former method, then somewhere within the model program the format (i.e. field specification) for data entry is defined for every number which must be read in this fashion.

The FORTRAN code of figure 2.3 directs a program to read five real numbers. The first three are read using a format specifier, whereas the last two are read in free field fashion.

```
      READ(20,100) A,B,C
100   FORMAT(3F10.0)
      READ(20,*) D,E
```

Figure 2.3 Formatted and free field input.

The relevant part of the input file may be as illustrated in figure 2.4.

```
       6.32  1.42E-05123.456789
34.567, 1.2E17
```

Figure 2.4 Numbers read using the code of figure 2.3.

Notice how no whitespace or comma is needed between numbers which are read using a field specifier. The format statement labelled "100" in figure 2.3 directs that variable A be read from the first 10 positions on the line, that variable B be read from the next 10 positions, and that variable C be read from the 10 positions thereafter. When the program reads any of these numbers it is unconcerned as to what characters lie outside of the field on which its attention is currently focussed. However the numbers to be read into variables D and E must be separated by whitespace or a comma in order that the program knows where one number ends and the next number begins.

Suppose all of variables A to E are model parameters, and that PEST has been assigned the task of estimating them. For convenience we provide the same names for these parameters as are used by the model code (this, of course, will not normally be the case). The template fragment corresponding to figure 2.4 may then be as set out in figure 2.5. Notice how the parameter space for each of parameters A, B and C is 10 characters wide, and that the parameter spaces abut each other in accordance with the expectations of the model as defined through the format specifier of figure 2.3. If the parameter space for any of these parameters is greater than 10 characters in width, then PEST, when it replaces each parameter space by the current parameter value, would construct a model input file which would be incorrectly read by the model. (You could have designed parameter spaces less than 10 characters wide if you wished, as long as you placed enough whitespace between each parameter space in order that the number which will replace each such space when PEST writes the model input file falls within the field expected by the model. However, defining the parameter spaces in this way would achieve nothing as there would be no advantage in using less than the full 10 characters allowed by the model.)

```
#  A      ##     B    ##  C      #
#   D        #, #  E         #
```

Figure 2.5 Fragment of a template file corresponding to parameters represented in figure 2.4.

Parameters D and E are treated very differently to parameters A, B and C. As figure 2.3 shows, the model simply expects two numbers in succession. If the spaces for parameters D and E appearing in figure 2.5 are replaced by two numbers (each will be 13 characters long) the model's requirement for two numbers in succession separated by whitespace or a comma will have been satisfied, as will PEST's preference for maximum precision.

Comparing figures 2.4 and 2.5, it is obvious that the spaces for parameters D and E on the

template file are greater than the spaces occupied by the corresponding numbers on the model input file from which the template file was constructed; the same applies for the parameter spaces defined in figure 2.2 pertaining to the model input file of figure 2.1. In most cases of template file construction, a model input file will be used as the starting point. In such a file, numbers read using free field input will often be written with trailing zeros omitted. In constructing the template file you should recognise which numbers are read using free field input and expand the parameter space (to the right) accordingly beyond the original number, making sure to leave whitespace or a comma between successive spaces, or between a parameter space and a neighbouring character or number.

Similarly, numbers read through field-specifying format statements may not occupy the full field width in a model input file from which a template file is being constructed (e.g. variable A in figure 2.4). In such a case you should, again, expand the parameter space beyond the extent of the number (normally to the left of the number only) until the space coincides with the field defined in the format specifier with which the model reads the number. (If you are not sure of this field because the model manual does not inform you of it or you do not have the model's source code, you will often, nevertheless, be able to make a pretty good guess as to what the field width is. As long as the parameter space you define does not transgress the bounds of the format-specified field, and the space is wide enough to allow discrimination between a parameter value and an incrementally-varied parameter value, this is good enough.)

2.2.6 How PEST Fills a Parameter Space with a Number

PEST writes as many significant figures to a parameter space as it can. It does this so that even if a parameter space must be small in order to satisfy the input field requirements of a model, there is still every chance that a parameter value can be distinguished from its incrementally-varied counterpart so as to allow proper derivatives calculation with respect to that parameter. Also, as has already been discussed, even though PEST adjusts its internal representation of a parameter value to the precision with which the model can read it so that PEST and the model are using the same number, in general more precision is better.

Two user-supplied control variables, PRECIS and DPOINT affect the manner in which PEST writes a parameter value to a parameter space. Both of these variables are provided to PEST through the PEST control file. PRECIS is a character variable which must be supplied as either "single" or "double". It determines whether single or double precision protocol is to be observed in writing parameter values; unless a parameter space is greater than 13 characters in width it has no bearing on the precision with which a parameter value is written to a model input file, as this is determined by the width of the parameter space. If PRECIS is set to "single", exponents are represented by the letter "e"; also if a parameter space is greater than 13 characters in width, only the last 13 spaces are used in writing the number representing the parameter value, any remaining characters within the parameter space being left blank. For the "double" alternative, up to 23 characters can be used to represent a number, and the letter "d" is used to represent exponents; also, extremely large and extremely small numbers can be represented.

If a model's input data fields are small, and there is nothing you can do about it, every effort must be made to "squeeze" as much precision as possible into the limited parameter spaces available. PEST will do this anyway, but it may be able to gain one or more extra significant figures if it does not need to include a decimal point in a number if the decimal point is redundant. Thus if a parameter space is 5 characters wide and the current value of the

parameter to which this field pertains is 10234.345, PEST will write the number as "1.0e4" or as "10234" depending on whether it must include the decimal point or not. Similarly, if the parameter space is 6 characters wide, the number 106857.34 can be represented as either "1.07e5" or "1069e2" depending on whether the decimal point must be included or not.

By assigning the string "nopoint" to the PEST control variable DPOINT, you can instruct PEST to omit the decimal point in the representation of a number if it can. However this should be done with great caution. If the model is written in FORTRAN and numbers are read using free field input, or using a field width specifier such as "(F6.0)" or "(E8.0)", the decimal point is not necessary. However in other cases the format specifier will insert its own decimal point (e.g. for specifiers such as "(F6.2)"), or enforce power-of-10 scaling (e.g. for specifiers such as "(E8.2)") if a decimal point is absent from an input number. Hence if you are unsure what to do, assign the string "point" to the control variable DPOINT; this will ensure that all numbers written to model input files will include a decimal point, thus overriding point-location or scaling conventions implicit in some FORTRAN format specifiers.

Note that if a parameter space is 13 characters wide or greater and PRECIS is set to "single", PEST will include the decimal point regardless of the setting of "DPOINT", for there are no gains to be made in precision through leaving it out. Similarly, if PRECIS is set to "double", no attempt is made to omit a decimal point if the parameter space is 23 characters wide or more.

Table 2.1 shows how the setting of DPOINT affects the representation of the number 12345.67. In examining this table, remember that PEST writes a number in such a way that the maximum possible precision is "squeezed" into each parameter space.

parameter space width (characters)	**DPOINT**	
	"point"	"nopoint"
8	12345.67	12345.67
7	12345.7	12345.7
6	12346.	12346.
5	1.2e4	12346
4	1.e4	12e3
3	***	1e4
2	**	**

Table 2.1 Representations of the number 12345.67

As explained below, a template file may contain multiple spaces for the same parameter. In such a case, PEST will write the parameter value to all those spaces using the minimum parameter space width specified for that particular parameter; for the wider spaces the number will be right-justified, with spaces padded on the left. In this way a consistent parameter value is written to all spaces pertaining to the one parameter.

2.2.7 Multiple Occurrences of the Same Parameter

Large numerical models which calculate the variation of some scalar or vector quantity over two or three-dimensional space may require on their input files large amounts of system property data written in the form of two- or three-dimensional arrays. For example, a finite-difference groundwater model may read arrays representing the distribution of hydraulic conductivity, storage coefficient, and other aquifer properties over the modelled area, each element within each array pertaining to one rectangular, finite-difference "cell". A finite-element model used in simulating geophysical traverse results over an orebody may require an array containing conductivity values for the various elements into which the orebody and surrounding earth are subdivided. For large grids or networks used to spatially discretise two- or three-dimensional inhomogeneous systems of this kind, hundreds, perhaps thousands, of numbers may be required to represent the distributed system properties.

If it is required that field measurements be used to infer system properties (using models such as these to link these properties to system response) certain assumptions regarding the variation in space of the distributed parameters must be made. A common assumption is that the model domain is "zoned". According to this assumption the system is subdivided into a number of areas or volumes in each of which a certain physical property is constant. Hence while the input arrays will still contain hundreds, maybe thousands, of elements, each element will be one of only *n* different numbers, where *n* is the number of zones into which the model domain has been subdivided.

It is a simple matter to construct a PEST template file for a model such as this. Firstly prepare for a model run in the usual way. Using the model preprocessor, assign *n* different values for a particular property to each of the *n* different model zones, writing the model input arrays to the model input files in the usual manner. Then, using the "search and replace" facility of a text editor, edit the model input file such that each occurrence within a particular array of the number representing the property of a certain zone is altered to a parameter space identifier (such as "# ro1 #"); remember to make the parameter space as wide as the model will allow in order to ensure maximum precision. If this is done in turn for each of the *n* different numbers occurring in the array, using a different parameter name in place of each different number, the array of numbers will have been replaced by an array of parameter spaces. When PEST writes the model input file it will, as usual, replace each such parameter space with the corresponding current parameter value; hence it will reconstruct an array containing hundreds, maybe thousands, of elements, but in which only *n* different numbers are represented.

The occurrence of multiple incidences of the same parameter is not restricted to the one file. If a model has multiple input files, and if a particular parameter which you would like to estimate appears on more than one of these files, then at least one space for this parameter will appear on more than one template file. PEST passes no judgement on the occurrence of parameters within template files or across template files. However it does require that each parameter cited in the PEST control file occur at least once on at least one template file, and that each parameter cited in a template file be provided with bounds and an initial value in the PEST control file.

(Note that more sophisticated means of spatially-distributed parameterisation than that provided by zones of piecewise constancy are available through programs such as PLPROC which are downloadable from the PEST web pages. Among other devices, PLPROC supports the use of pilot points. At the time of writing, interpolation from these points to a model grid or mesh can be implemented using kriging, inverse power of distance interpolation and radial

basis functions.)

2.2.8 Preparing a Template File

Preparation of a template file is a simple procedure. For most models it can be done in a matter of moments using a text editor to replace parameter values on a typical model input file by their respective parameter space identifiers.

Once a template file has been prepared, it can be checked for correctness using the utility program TEMPCHEK; see part II of this manual. TEMPCHEK also has the ability to write a model input file on the basis of a template file and a user-supplied list of parameter values. If you then run your model, providing it with such a TEMPCHEK-prepared input file, you can verify that the model will have no difficulties in reading input files prepared by PEST.

2.3 Instruction Files

Of the possibly voluminous amounts of information that a model may write to its output file(s), PEST is interested in only a few numbers. These can be numbers for which corresponding field or laboratory data are available and for which the discrepancies between model output and measured values must be reduced to a minimum in the weighted least squares sense. Alternatively they can be model predictions of particular interest, or they can simply be model outputs for which sensitivities with respect to parameters are required. These particular model-generated numbers are referred to as "observations" or "model-generated observations" in the discussion which follows. Meanwhile, field or laboratory measurements of which they are the model-generated counterparts are referred to as "measurements".

For every model output file containing observations, you must provide an instruction file containing the directions which PEST must follow in order to read that file. Note that if a model output file is more than 2000 characters in width, PEST will be unable to read it; however a model output file can be of any length.

Some models write some or all of their output data to the terminal. You can redirect this screen output to a file using the ">" symbol and teach PEST how to read this file using a matching instruction file in the usual manner.

It is suggested that instruction files be provided with the extension "*.ins*" in order to distinguish them from other types of file.

2.3.1 Precision in Model Output Files

If there are any control variables which allow you to vary the precision with which a model's output data are written, these should be adjusted such that model outputs which are read by PEST are recorded with maximum available precision by the model. Unlike parameter values, for which precision is important but not essential, precision in the representation of model-generated observations is crucial. As was mentioned in the introduction, and as is discussed extensively in Doherty (2015), inversion and uncertainty quantification methods implemented by PEST require integrity of calculation of derivatives of model outputs with respect to model parameters. PEST calculates these derivatives using finite-difference techniques in which model-generated numbers of simular magnitude are subtracted from each other. Subtraction involves loss of precision if differences are small. Unless the numbers which PEST reads from model output files are represented with maximum precision on those files, loss of precision incurred through subtraction may be sufficient to invalidate thus-

calculated derivatives. As far as PEST is concerned, this is a recipe for numerical disaster.

2.3.2 How PEST Reads a Model Output File

PEST must be taught how to read a model output file and identify the numbers it must extract from that file. For this to happen, model output files read by PEST must be text files; PEST cannot read a binary file. If your model produces only binary files, you will need to write a simple program which reads this binary data and rewrites it in ASCII form; PEST can then search the ASCII file for the numbers it needs.

Unfortunately, a number cannot be read from model output files using the template concept. This is because many models cannot be relied upon to produce an output file of identical structure on each model run. For example, a model which calculates the underground stress regime in the vicinity of a tunnel may employ an iterative numerical solution scheme for which different numbers of iterations are required to achieve numerical convergence depending on the boundary conditions and material properties supplied for a particular run. If the model records on its output file the convergence history of the solution process, and the results of its stress calculations are recorded on the lines following this, the latter may be displaced downwards depending on the number of iterations required to calculate them.

So instead of using an output file template, you must provide PEST with a list of instructions on how to find observations on an output file. Basically, PEST finds observations on a model output file in the same way that a person does. A person runs his/her eye down the file looking for something which he/she recognises - a "marker"; if this marker is properly selected, observations can usually be linked to it in a simple manner. For example, if you are looking for the outcome of the above stress model's deliberations at an elapsed time of 100 days, you may instruct PEST to read its output file looking for the marker

```
STRESS CALCULATED AT FINITE ELEMENT NODES: ELAPSED TIME = 100 DAYS
```

A particular outcome for which you have a corresponding field measurement may then be found, for example, between character positions 23 and 30 on the 4th line following the above marker, or as the 5th item on the 3rd line after the marker, etc. Note that for simple models, especially "home-made", single-purpose models where little development time has been invested in highly descriptive output files, no markers may be necessary, the default initial marker being the top of the file.

Markers can be of either primary or secondary type. PEST uses a primary marker as it scans the model output file line by line, looking for a reference point for subsequent observation identification or further scanning. A secondary marker is used for a reference point as a single line is examined from left to right.

2.3.3 An Example Instruction File

Figure 2.6 shows an output file written by the model whose input file appears in figure 2.1. Suppose that we wish to estimate the parameters appearing in the template file of figure 2.2 (i.e. the resistivities of the three half-space layers and the thicknesses of the upper two) by comparing apparent resistivities generated by the model with a set of apparent resistivities provided by field measurement. Then we need to provide instructions to PEST on how to read each of the apparent resistivities appearing in figure 2.6. An appropriate instruction file is shown in figure 2.7.

```
SCHLUMBERGER ELECTRIC SOUNDING

Apparent resistivities calculated using the linear filter method

electrode spacing  apparent resistivity
    1.00            1.21072
    1.47            1.51313
    2.15            2.07536
    3.16            2.95097
    4.64            4.19023
    6.81            5.87513
    10.0            8.08115
    14.7            10.8029
    21.5            13.8229
    31.6            16.5158
    46.4            17.7689
    68.1            16.4943
    100.            12.8532
    147.            8.79979
    215.            6.30746
    316.            5.40524
    464.            5.15234
    681.            5.06595
   1000.            5.02980
```

Figure 2.6 A model output file.

```
pif @
@electrode@
l1 [ar1]21:27
l1 [ar2]21:27
l1 [ar3]21:27
l1 [ar4]21:27
l1 [ar5]21:27
l1 [ar6]21:27
l1 [ar7]21:27
l1 [ar8]21:27
l1 [ar9]21:27
l1 [ar10]21:27
l1 [ar11]21:27
l1 [ar12]21:27
l1 [ar13]21:27
l1 [ar14]21:27
l1 [ar15]21:27
l1 [ar16]21:27
l1 [ar17]21:27
l1 [ar18]21:27
l1 [ar19]21:27
```

Figure 2.7 A PEST instruction file.

2.3.4 The Marker Delimiter

The first line of a PEST instruction file must begin with the three letters "pif" which stand for "PEST instruction file". Then, after a single space, must follow a single character, the marker delimiter. The role of the marker delimiter in an instruction file is not unlike that of the parameter delimiter in a template file. Its role is to define the extent of a marker; a marker delimiter must be placed just before the first character of a text string comprising a marker and immediately after the last character of the marker string. In treating the text between a

pair of marker delimiters as a marker, PEST does not try to interpret this text as a list of instructions.

You can choose the marker delimiter character yourself; however your choice is limited. A marker delimiter must not be one of the characters A - Z, a - z, 0 - 9, !, [,], (,), :, the space or tab characters, or &; the choice of any of these characters may result in confusion, as they may occur elsewhere in an instruction file in a role other than that of marker delimiter. Note that the character you choose as the marker delimiter should not occur within the text of any markers as this, too, will cause confusion.

2.3.5 Observation Names

In the same way that each parameter must have a unique name, so too must each observation be provided with a unique name. Observation names must be twenty characters or less in length. These twenty characters can be any ASCII characters except for [,], (,), or the marker delimiter character.

As discussed above, a parameter name can occur more than once within a parameter template file; PEST simply replaces each parameter space in which the name appears with the current value of the pertinent parameter. However the same does not apply to an observation name. Every observation is unique and must have a unique observation name. In figure 2.6, observations are named “ar1”, “ar2” etc. These same observation names must also be cited in the PEST control file where measurement values and weights are provided.

There is one observation name, however, to which these rules do not apply, namely the dummy observation name “dum”. This name can occur many times, if necessary, in an instruction file; it signifies to PEST that, although the observation is to be located as if it were a normal observation, the number corresponding to the dummy observation on the model output file is not actually matched with any laboratory or field measurement. Hence an observation named “dum” must not appear in the PEST control file where measurement values are provided and observation weights are assigned. As is illustrated below, the dummy observation is simply a device for model output file navigation.

2.3.6 The Instruction Set

Each of the available PEST instructions is now described in detail. When creating your own instruction files, the syntax provided for each instruction must be followed exactly. If a number of instruction items appear on a single line of an instruction file, these items must be separated from each other by at least one space. Instructions pertaining to a single line on a model output file are written on a single line of a PEST instruction file. Thus the start of a new instruction line signifies that PEST must read at least one new model output file line; just how many lines it needs to read depends on the first instruction on the new instruction line. Note, however, that if the first instruction on the new line is the character “&”, the new instruction line is simply a continuation of the old one. Like all other instruction items, the “&” character used in this context must be separated from its following instruction item by at least one space.

PEST reads a model output file in the forward (top-to-bottom) direction, looking to the instructions in the instruction file to tell it what to do next. Instructions should be written with this in mind; an instruction cannot direct PEST to “backtrack” to a previous line on the model output file. Also, because PEST processes model output file lines from left to right, an instruction cannot direct PEST backwards to an earlier part of a model output file line than the part of the line to which its attention is currently focussed as a result of the previous

instruction.

Primary Marker

Unless it is a continuation of a previous line, each instruction line must begin with either of two instruction items, namely a primary marker or a line advance item. The primary marker has already been discussed briefly. It is a string of characters, bracketed at each end by a marker delimiter. If a marker is the first item on an instruction line, then it is a primary marker; if it occurs later in the line, following other instruction items, it is a secondary marker, the operation of which will be discussed below.

On encountering a primary marker in an instruction file PEST reads the model output file, line by line, searching for the string between the marker delimiter characters. When it finds the string it places its "cursor" at the last character of the string. (Note that this cursor is never actually seen by the PEST user; it simply marks the point where PEST is at in its processing of the model output file.) This means that if any further instructions on the same instruction line as the primary marker direct PEST to further processing of this line, that processing must pertain to parts of the model output file line following the string identified as the primary marker.

Note that if there are blank characters in a primary (or secondary) marker, exactly the same number of blank characters is expected in the matching string on the model output file.

Often, as in figure 2.7, a primary marker will be part or all of some kind of header or label; such a header or label often precedes a model's listing of the outcomes of its calculations and thus makes a convenient reference point from which to search for the latter. It should be noted, however, that the search for a primary marker is a time-consuming process as each line of the model output file must be individually read and scanned for the marker. Hence if the same observations are always written to the same lines of a model output file (these lines being invariant from model run to model run), you should use the line advance item in preference to a primary marker.

A primary marker may be the only item on a PEST instruction line, or it may precede a number of other items directing further processing of the line containing the marker. In the former case the purpose of the primary marker is simply to establish a reference point for further downward movement within the model output file as set out in subsequent instruction lines.

Primary markers can provide a useful means of navigating a model output file. Consider the extract from a model output file shown in figure 2.8 (the dots replace one or a number of lines not shown in the example in order to conserve space). The instruction file extract shown in figure 2.9 provides a means to read the numbers comprising the third solution vector. Notice how the "SOLUTION VECTOR" primary marker is preceded by the "PERIOD NO. 3" primary marker. The latter marker is used purely to establish a reference point from which a search can be made for the "SOLUTION VECTOR" marker; if this reference point were not established (using either a primary marker or line advance item) PEST would read the solution vector pertaining to a previous time period.

```
            .
            .
TIME PERIOD NO. 1 --->
            .
            .
SOLUTION VECTOR:
  1.43253  6.43235  7.44532  4.23443  91.3425  3.39872
            .
            .
TIME PERIOD NO. 2 --->
            .
            .
SOLUTION VECTOR
  1.34356  7.59892  8.54195  5.32094  80.9443  5.49399
            .
            .
TIME PERIOD NO. 3 --->
            .
            .
SOLUTION VECTOR
  2.09485  8.49021  9.39382  6.39920  79.9482  6.20983
```

Figure 2.8 Extract from a model output file.

```
pif *
      .
      .
*PERIOD NO. 3*
*SOLUTION VECTOR*
l1 (obs1)5:10 (obs2)12:17 (obs3)21:28 (obs4)32:37 (obs5)41:45
& (obs6)50:55
      .
      .
```

Figure 2.9 Extract from an instruction file.

Line Advance

The syntax for the line advance item is "l*n*" where *n* is the number of lines to advance; note that "l" is "el", the twelfth letter of the alphabet, not "one". The line advance item must be the first item of an instruction line; it and the primary marker are the only two instruction items which can occupy this initial spot. As was explained above, the initial item in an instruction line is always a directive to PEST to move at least one line further in its perusal of the model output file (unless it is a continuation character). In the case of the primary marker, PEST stops reading new lines when it finds the pertinent text string. However in implementing a line advance PEST does not need to examine model output file lines as it advances. It simply moves forward *n* lines, placing its processing cursor just before the beginning of this new line, this point becoming the new reference point for further processing of the model output file.

Normally a line advance item is followed by other instructions. However if the line advance item is the only item on an instruction line this does not break any syntax rules.

In figure 2.6 model-calculated apparent resistivities are written on consecutive lines. Hence before reading each observation, PEST is instructed to move to the beginning of a new line using the "l1" line advance item; see figure 2.7.

If a line advance item leads the first instruction line of a PEST instruction file, the reference

point for line advance is taken as a "dummy" line just above the first line of the model output file. Thus if the first instruction line begins with "l1", processing of the model output file begins on its first line; similarly, if the first instruction line begins with "l8", processing of the model output file begins at its eighth line.

Secondary Marker

A secondary marker is a marker which does not occupy the first position of a PEST instruction line. Hence it does not direct PEST to move downwards on the model output file (though it can be instrumental in this - see below); rather it instructs PEST to move its cursor along the current model output file line until it finds the secondary marker string, and to place its cursor on the last character of that string ready for subsequent processing of that line.

Figure 2.10 shows an extract from a model output file while figure 2.11 shows the instructions necessary to read the potassium concentration from this output file. A primary marker is used to place the PEST cursor on the line above that on which the calculated concentrations are recorded for the distance in which we are interested. Then PEST is directed to advance one line and read the number following the "K:" string in order to find an observation named "kc"; the exclamation marks surrounding "kc" will be discussed shortly.

```
            .
            .
DISTANCE = 20.0: CATION CONCENTRATIONS:-
Na: 3.49868E-2  Mg: 5.987638E-2  K: 9.987362E-3
            .
            .
```

Figure 2.10 Extract from a model output file.

```
pif ~
      .
      .
~DISTANCE = 20.0~
l1 ~K:~ !kc!
      .
      .
```

Figure 2.11 Extract from an instruction file.

A useful feature of the secondary marker is illustrated in figures 2.12 and 2.13 which represent a model output file extract and a corresponding instruction file extract, respectively. If a particular secondary marker is preceded only by other markers (including, perhaps, one or a number of secondary markers and certainly a primary marker), and the text string corresponding to that secondary marker is not found on a model output file line on which the previous markers' strings have been located, PEST will assume that it has not yet found the correct model output line and resume its search for a line which holds the text from all three markers. Thus the instruction "%TIME STEP 10%" will cause PEST to pause on its downward journey through the model output file at the first line illustrated in figure 2.12. However, when it does not find the string "STRAIN" on the same line it re-commences its perusal of the model output file, looking for the string "TIME STEP 10" again. Eventually it finds a line containing both the primary and secondary markers and, having done so, commences execution of the next instruction line.

```
            .
            .
TIME STEP 10 (13 ITERATIONS REQUIRED) STRESS --->
X = 1.05 STRESS = 4.35678E+03
X = 1.10 STRESS = 4.39532E+03
            .
            .
TIME STEP 10 (BACK SUBSTITUTION) STRAIN --->
X = 1.05 STRAIN = 2.56785E-03
X = 1.10 STRAIN = 2.34564E-03
            .
            .
```

Figure 2.12 Extract from a model output file.

It is important to note that if any instruction items other than markers precede an unmatched secondary marker, PEST will assume that the mismatch is an error condition and cease execution with an appropriate error message. Note also that secondary markers may be used sequentially. For example if the STRAIN variable is always in position 2, then the pertinent line in the instruction file of figure 2.13 could be replaced by "l1 %=% %=% !str1!". This is handy for comma-delimited output files.

```
pif %
      .
      .
%TIME STEP 10% %STRAIN%
l1 %STRAIN =% !str1!
l1 %STRAIN =% !str2!
      .
      .
```

Figure 2.13 Extract from an instruction file.

Whitespace

The whitespace instruction is similar to the secondary marker in that it allows the user to navigate through a model output file line prior to reading a non-fixed observation (see below). It directs PEST to move its cursor forwards from its current position until it encounters the next blank character. PEST then moves the cursor forward again until it finds a nonblank character, finally placing the cursor on the blank character preceding this nonblank character (i.e. on the last blank character in a sequence of blank characters) ready for the next instruction. The whitespace instruction is a simple “w”, separated from its neighbouring instructions by at least one blank space.

Consider the model output file line represented below.

```
MODEL OUTPUTS:  2.89988  4.487892  -4.59098   8.394843
```

The following instruction line directs PEST to read the fourth number on the above line.

```
%MODEL OUTPUTS:% w w w w !obs1!
```

The instruction line begins with a primary marker, allowing PEST to locate the above line on the model output file. After this marker is processed the PEST cursor rests on the “:” character of “OUTPUTS:”, i.e. on the last character of the marker string. In response to the first whitespace instruction PEST finds the next whitespace and then moves its cursor to the end of this whitespace, i.e. just before the “2” of the first number on the above model output file line. The second whitespace instruction moves the cursor to the blank character preceding the first “4” of the second number on the above line; processing of the third whitespace

instruction results in PEST moving its cursor to the blank character just before the negative sign. After the fourth whitespace instruction is implemented, the cursor rests on the blank character preceding the last number; the latter can then be read as a non-fixed observation (see below).

Tab

The tab instruction places the PEST cursor at a user-specified character position (i.e. column number) on the model output file line which PEST is currently processing. The instruction syntax is "tn" where *n* is the column number. The column number is obtained by counting character positions (including blank characters) from the left side of any line, starting at 1. Like the whitespace instruction, the tab instruction can be useful in navigating through a model output file line prior to locating and reading a non-fixed observation. For example, consider the following line from a model output file:

```
TIME(1): A = 1.34564E-04, TIME(2): A = 1.45654E-04, TIME(3): A = 1.54982E-04
```

The value of A at TIME(3) could be read using the instruction line:

```
l4 t60 %=% !a3!
```

Here it is assumed that PEST was previously processing the fourth line prior to the above line in the model output file; the marker delimiter character is assumed to be "%". Implementation of the "t60" instruction places the cursor on the ":" following the "TIME(3)" string, for the colon is in the sixtieth character position of the above line. PEST is then directed to find the next "=" character. From there it can read the last number on the above line as a non-fixed observation (see below).

Fixed Observations

An observation reference can never be the first item on an instruction line; either a primary marker or line advance item must come first in order to place PEST's cursor on the line on which one or more observations may lie. If there is more than one observation on a particular line of the model output file, these observations must be read from left to right, backward movement along any line being disallowed.

Observations can be identified in one of three ways. The first way is to tell PEST that a particular observation can be found between, and including, columns n_1 and n_2 on the model output file line on which its cursor is currently resting. This is by far the most efficient way to read an observation value because PEST does not need to do any searching; it simply reads a number from the space identified. Observations read in this way are referred to as "fixed observations".

Figure 2.14 shows how the numbers listed in the third solution vector of figure 2.8 can be read as fixed observations. The instruction item informing PEST how to read a fixed observation consists of two parts. The first part consists of the observation name enclosed in square brackets, while the second part consists of the first and last columns from which to read the observation. Note that no space must separate these two parts of the observation instruction; PEST always construes a space in an instruction file as marking the end of one instruction item and the beginning of another (unless the space lies between marker delimiters).

```
pif *
      .
      .
*PERIOD NO. 3*
*SOLUTION VECTOR*
l1 [obs1]1:9 [obs2]10:18 [obs3]19:27 [obs4]28:36 [obs5]37:45
& [obs6]46:54
      .
      .
```

Figure 2.14 Extract from an instruction file.

Reading numbers as fixed observations is useful when the model writes its output in tabular form using fixed field width specifiers. However you must be very careful when specifying the column numbers from which to read the number. The space defined by these column numbers must be wide enough to accommodate the maximum length that the number will occupy over the many model runs for which PEST will read the model's output file; if it is not wide enough, PEST may read only a truncated part of the number or omit a negative sign preceding the number. However the space must not be so wide that it includes part of another number; in this case a run-time error will occur and PEST will terminate execution with an appropriate error message.

Where a model writes its results in the form of an array of numbers, it is not an uncommon occurrence for these numbers to abut each other. Consider, for example, the following FORTRAN code fragment.

```
      A=1236.567
      B=8495.0
      C=-900.0
      WRITE(10,20) A,B,C
20    FORMAT(3(F8.3))
```

The result is

```
1236.5678495.000-900.000
```

In this case there is no choice but to read these numbers as fixed observations. (Both of the alternative ways to read an observation require that the observation be surrounded by either whitespace or a string that is invariant from model run to model run and can thus be used as a marker.) Hence to read the above three numbers as observations A, B and C the following instruction line may be used.

```
l1 [A]1:8 [B]9:16 [C]17:24
```

If an instruction line contains only fixed observations there is no need for it to contain any whitespace or tabs; nor will there be any need for a secondary marker, (unless the secondary marker is being used in conjunction with a primary marker in determining which model output file line the PEST cursor should settle on - see above). This is because these items are normally used for navigating through a model output file line prior to reading a non-fixed observation (see below); such navigation is not required to locate a fixed observation as its location on a model output file line is defined without ambiguity by the column numbers included within the fixed observation instruction.

Semi-Fixed Observations

Figure 2.9 demonstrates the use of semi-fixed observations. Semi-fixed observations are similar to fixed observations in that two numbers are provided in the pertinent instruction

item, the purpose of these numbers being to locate the observation's position by column number on the model output file. However, in contrast to fixed observations, these numbers do not locate the observation exactly. When PEST encounters a semi-fixed observation instruction it proceeds to the first of the two nominated column numbers and then, if this column is not occupied by a non-blank character, it searches the output file line from left to right beginning at this column number, until it reaches either the second identified column or a non-blank character. If it reaches the second column before finding a non-blank character, an error condition arises. However if it finds a non-blank character at the first of the two column numbers, it then locates the nearest whitespace on either side of the character; in this way, it identifies one or a number of non-blank characters sandwiched between whitespace ("whitespace" includes the beginning and/or the end of the model output file line). It tries to read these characters as a number, this number being the value of the observation named in the semi-fixed observation instruction. Obviously the width of this number can be greater than the difference between the column numbers cited in the semi-fixed observation instruction.

Like a fixed observation, the instruction to read a semi-fixed observation consists of two parts, namely the observation name followed by two column numbers, the latter being separated by a colon; the column numbers must be in ascending order. However for semi-fixed observations, the observation name is enclosed in round brackets rather than square brackets. Again, there must be no space separating the two parts of the semi-fixed observation instruction.

Reading a number as a semi-fixed observation is useful if you are unsure how large the representation of the number could stretch on a model output file as its magnitude grows and/or diminishes in PEST-controlled model runs; it is also useful if you do not know whether the number is left or right justified. However you must be sure that at least part of the number will always fall between (and including) the two nominated columns and that, whenever the number is written and whatever its size, it will always be surrounded either by whitespace or by the beginning or end of the model output file line. If, when reading the model output file, PEST encounters only whitespace between (and including) the two nominated column numbers, or if it encounters non-numeric characters or two number fragments separated by whitespace, an error condition will occur and PEST will terminate execution with an appropriate error message.

As for fixed observations, it is normally not necessary to have secondary markers, whitespace and tabs on the same line as a semi-fixed observation, because the column numbers provided with the semi-fixed observation instruction determine the location of the observation on the line. As always, observations must be read from left to right on any one instruction line; hence if more than one semi-fixed observation instruction is provided on a single PEST instruction line, the column numbers pertaining to these observations must increase from left to right.

For the case illustrated in figures 2.6 and 2.7, all the fixed observations could have been read as semi-fixed observations, with the difference between the column numbers either remaining the same or being reduced to substantially smaller than that shown in figure 2.7. However it should be noted that it takes more effort for PEST to read a semi-fixed observation than it does for it to read a fixed observation as PEST must establish for itself the extent of the number that it must read.

After PEST has read a semi-fixed observation its cursor resides at the end of the number which it has just read. Any further processing of the line must take place to the right of that

position.

Non-Fixed Observations

Figures 2.11 and 2.13 demonstrate the use of non-fixed observations. A non-fixed observation instruction does not include any column numbers because the number which PEST must read is found using secondary markers and/or other navigational aids such as whitespace and tabs which precede the non-fixed observation on the instruction line.

If you do not know exactly where, on a particular model output file line, a model will write the number corresponding to a particular observation, but you do know the structure of that line, then you can use this knowledge to navigate your way to the number. In the PEST instruction file, a non-fixed observation is represented simply by the name of the observation surrounded by exclamation marks; as usual, no spaces should separate the exclamation marks from the observation name as PEST interprets spaces in an instruction file as denoting the end of one instruction item and the beginning of another.

When PEST encounters a non-fixed observation instruction it first searches forward from its current cursor position until it finds a non-blank character; PEST assumes this character is the beginning of the number representing the non-fixed observation. Then PEST searches forward again until it finds either a blank character, the end of the line, or the first character of a secondary marker which follows the non-fixed observation instruction in the instruction file; PEST assumes that the number representing the non-fixed observation finishes at the previous character position. In this way it identifies a string of characters which it tries to read as a number; if it is unsuccessful in reading a number because of the presence of non-numeric characters or some other problem, PEST terminates execution with a run-time error message. A run time error message will also occur if PEST encounters the end of a line while looking for the beginning of a non-fixed observation.

Consider the output file fragment shown in figure 2.15. The species populations at different times cannot be read as either fixed or semi-fixed observations because the numbers representing these populations cannot be guaranteed to fall within a certain range of column numbers on the model output file because “iterative adjustment” may be required in the calculation of any such population. Hence we must find our way to the number using another method; one such method is illustrated in figure 2.16.

```
             .
             .
SPECIES POPULATION AFTER 1 YEAR  = 1.23498E5
SPECIES POPULATION AFTER 2 YEARS = 1.58374E5
SPECIES POPULATION AFTER 3 YEARS (ITERATIVE ADJUSTMENT REQUIRED)= 1.78434E5
SPECIES POPULATION AFTER 4 YEARS = 2.34563E5
             .
             .
```

Figure 2.15 Extract from a model output file.

```
pif *
      .
      .
*SPECIES* *=* !sp1!
l1 *=* !sp2!
l1 *=* !sp3!
l1 *=* !sp4!
      .
      .
```

Figure 2.16 Extract from an instruction file.

A primary marker is used to move the PEST cursor to the first of the lines shown in figure 2.15. Then, noting that the number representing the species population always follows a "=" character, the "=" character is used as a secondary marker. After it processes a secondary marker, the PEST cursor always resides on the last character of that marker, in this case on the "=" character itself. Hence after reading the "=" character, PEST is able to process the !sp1! instruction by isolating the string "1.23498E5" in the manner described above.

After it reads the model-calculated value for observation "sp1", PEST moves to the next instruction line. In accordance with the "l1" instruction, PEST reads into its memory the next line of the model output file. It then searches for a "=" character and reads the number following this character as observation "sp2". This procedure is then repeated for observations "sp3" and "sp4".

Successful identification of a non-fixed observation depends on the instructions preceding it. The secondary marker, tab and whitespace instructions will be most useful in this regard, though fixed and semi-fixed observations may also precede a non-fixed observation; remember that in all these cases PEST places its cursor over the last character of the string or number it identifies on the model output file corresponding to an instruction item, before proceeding to the next instruction.

Consider the model output file line shown below as a further illustration of the use of non-fixed observations.

```
4.33 -20.3 23.392093 3.394382
```

If we are interested in the fourth of these numbers but we are unsure as to whether the numbers preceding it might not be written with greater precision in some model runs (hence pushing the number in which we are interested to the right), then we have no alternative but to read the number as a non-fixed observation. However if the previous numbers vary from model run to model run, we cannot use a secondary marker either; nor can a tab be used. Fortunately, whitespace comes to the rescue, with the following instruction line taking PEST to the fourth number:

```
l10 w w w !obs1!
```

Here it is assumed that, prior to reading this instruction, the PEST cursor was located on the 10th preceding line of the model output file. As long as we can be sure that no whitespace will ever precede the first number, there will always be three incidences of whitespace preceding the number in which we are interested. However, if it happens that whitespace may precede the first number on some occasions, while on other occasions it may not, then we can read the first number as a dummy observation as shown below:

```
l10 !dum! w w w !obs1!
```

As was explained previously, the number on the model output file corresponding to an

observation named “dum” is not actually used; nor can the name “dum” appear in the “observation data” section of the PEST control file. The use of this name is reserved for instances like the present case where a number must be read in order to facilitate navigation along a particular line of the model output file. The number is read according to the non-fixed observation protocol, for only observations of this type can be dummy observations.

An alternative to the use of whitespace in locating the observation “obs1” in the above example could involve using the dummy observation more than once. Hence the instruction line below would also enable the number representing “obs1” to be located and read.

```
l10 !dum! !dum! !dum! !obs1!
```

However had the numbers in the above example been separated by commas instead of whitespace, the commas should have been used as secondary markers in order to find “obs1”.

A number not surrounded by whitespace can still be read as a non-fixed observation with the proper choice of secondary markers. Consider the model output file line shown below.

```
SOIL WATER CONTENT (NO CORRECTION)=21.345634%
```

It may not be possible to read the soil water content as a fixed observation because the “(NO CORRECTION)” string may or may not be present after any particular model run. Reading it as a non-fixed observation appears troublesome as the number is neither preceded nor followed by whitespace. However a suitable instruction line is

```
l5 *=* !sws! *%*
```

Notice how a secondary marker (i.e. *%*) is referenced even though it occurs after the observation we wish to read. If this marker were not present, a run-time error would occur when PEST tries to read the soil water content because it would define the observation string to include the “%” character and, naturally, would be unable to read a number from a string which includes non-numeric characters. However by including the “%” character as a secondary marker after the number representing the observation “sws”, PEST is instructed to separate the character from the string before trying to read the number. But note that if a post-observation secondary marker of this type begins with a numerical character, PEST cannot be guaranteed not to include this character with the number representing the value of the observation if there is no whitespace separating it from the observation.

The fact that there is no whitespace between the “=” character and the number we wish to read causes PEST no problems either. After processing of the “=” character as a secondary marker, the PEST processing cursor falls on the “=” character itself. The search for the first non-blank character initiated by the !sws! instruction terminates on the very next character after “=”, i.e. the “2” character. PEST then accepts this character as the left boundary of the string from which it must read the soil moisture content and searches forwards for the right boundary of the string in the usual manner.

After PEST has read a non-fixed observation, it places its cursor on the last character of the observation number. It can then undertake further processing of the model output file line to read further non-fixed, fixed or semi-fixed observations, or process navigational instructions as directed.

Continuation

You can break an instruction line between any two instructions by using the continuation character, “&”, to inform PEST that a certain instruction line is actually a continuation of the previous line. Thus the instruction file fragment

```
l1 %RESULTS% %TIME (4)% %=% !obs1! !obs2! !obs3!
```

is equivalent to

```
l1
& %RESULTS%
& %TIME (4)%
& %=%
& !obs1!
& !obs2!
& !obs3!
```

For both these fragments, the marker delimiter is assumed to be "%". Note that the continuation character must be separated from the instruction which follows it by at least one space.

2.3.7 Making an Instruction File

An instruction file can be built using a text editor. Alternatively it can be written by software dedicated to this purpose such as a model graphical user interface which supports PEST, or members of the Groundwater and Surface Water Utilities downloadable from the PEST web pages.

Caution must always be exercised in building an instruction set to read a model output file, especially if navigational instructions such as markers, whitespace, tabs and dummy observations are used. PEST will always follow your instructions to the letter, but it may not read the number you intend it to read if you get an instruction wrong. If PEST tries to read an observation but does not find a number where it expects to find one, a run-time error will occur. PEST will inform you of where it encountered the error, and of the instruction it was implementing when the error occurred; this should allow you to find the problem. However if PEST actually reads the wrong number from the model output file, this may only become apparent if an unusually high objective function results, or if PEST is unable to lower the objective function on successive optimisation iterations. Alternatively if the number which PEST is instructed to read is a model prediction, or if PEST is being asked purely to compute sensitivities of this number to model parameters, the error may never be apparent. If in doubt, check PEST-generated files for numbers that PEST reads from model output files to make sure that they meet expectations.

Included in the PEST suite are two programs which can be used to verify that instruction files have been built correctly. Program PESTCHEK, when checking all PEST input data for errors and inconsistencies prior to a PEST run, reads all the instruction files cited in a PEST control file ensuring that no syntax errors are present in any of these files. Program INSCHEK, on the other hand, checks a single PEST instruction file for syntax errors. If an instruction file is error-free, INSCHEK can then use that instruction file to read a model output file, printing out a list of observation values read from that file. In this way you can be sure that your instruction set "works" before it is actually used by PEST.

3. What PEST Does

3.1 General

This chapter provides an overview of what PEST does. A general understanding of its operations is required before its operational details can be explained, and before descriptions can be provided of ways in which you can alter its operational details to suit your own modelling and inversion purposes.

All of the theory on which PEST and its utility support software is based is presented in Doherty (2015) (i.e. the "PEST book") and will not be repeated in this manual. However, where necessary, equations from the PEST book will be cited herein so that you can easily relate aspects of PEST's operations to the theory which supports them.

Where applicable, variables which control the way in which PEST performs will be mentioned in the following overview. These variables reside in the PEST control file. While later chapters will describe this file in detail, it is introduced below to set a context for the subject matter of the present chapter.

As was discussed in the introduction to this text, utility software which supports and compliments the use of PEST is documented in part II of this manual. Part I of the PEST manual (i.e. the document that you are reading right now) describes PEST itself, the global SCEUA_P and CMAES_P optimisers, and the very basic SENSAN sensitivity analyser.

3.2 The PEST Control File

3.2.1 General

PEST requires three types of input file. Two of these were discussed in the previous chapter, namely template and instruction files. For any particular PEST run, as many of each of these must be provided as there are model input files on which parameters reside and model output files from which numbers must be read, respectively. However there is only one PEST control file.

Specifications of the PEST control file, together with the names of all variables which can appear in this file, are provided in appendix A of this text and discussed in detail in chapter 5. The PEST control file is divided into sections. Some of these sections are optional. Some of the variables which reside in some of these sections are optional. Optional variables are enclosed in square brackets in figure A1.1 of appendix A.

Numbers and text strings appearing in the PEST control file which provide the values of PEST control variables must be separated by one or more spaces. They are read by PEST using free-field formatting. All text is case insensitive, whether it denotes a section header or the value of a control variable.

The first line of a PEST control file must contain only the characters "pcf", this standing for "PEST control file".

3.2.2 Sections of the PEST Control File

Each section of the PEST control file must begin with a single line of text comprising the header to that section. This line must begin with a "*" character and must be immediately

followed by a space. The text which follows these leading characters must be provided exactly as shown in figure A1.1.

The sections that are featured in a PEST, control file and the roles that these sections play, are provided in table 3.1. Sections must appear in the same order as that in which they appear in this table.

Section	**Optional/ Mandatory**	**Contents**
control data	mandatory	problem dimensions, mode of PEST operation, termination criteria, change limits and other control data
automatic user intervention	optional	variables which govern operation of PEST's automatic user intervention functionality
singular value decomposition	optional	variables which govern operation of singular value decomposition when used as an inverse problem solution mechanism
lsqr	optional	variables which govern operation of the LSQR algorithm when used as an inverse problem solution mechanism
sensitivity reuse	optional	variables which determine whether and how PEST re-uses some parameter sensitivities in subsequent iterations
svd assist	optional	variables which specify the manner in which the SVD-assist methodology is employed in solving an inverse problem
parameter groups	mandatory	variables which govern the way in which finite-difference derivatives are calculated
parameter data	mandatory	parameter initial values, transformation status, bounds, groups, scales and offsets
observation groups	mandatory	lists observation groups, and provides names of files holding observation covariance matrices
observation data	mandatory	provides measured values of observations, weights and groups to which observations belong
derivatives command line	optional	provides the command used to run the model if it calculates some of its own derivatives, and the file from which these derivatives will be read
model command line	mandatory	provides one or a number of commands used by PEST to run the model
model input/output	mandatory	lists template and corresponding model input files as well as instruction and corresponding model output files
prior information	optional	provides linear prior information employed in the inversion process
predictive analysis	optional	contains variables which govern PEST's operation when it is run in "predictive analysis" mode

regularisation	optional	contains variables which govern PEST's operation when it is run in "regularisation" mode
pareto	optional	contains variables which govern PEST's operation when it is run in "pareto" mode

Table 3.1 Sections of a PEST control file.

3.2.3 Naming Conventions

The PEST control file must have an extension of "*.pst*". You choose the filename base yourself. Let "*case*" represent your choice of filename base. Then the PEST control file is named *case.pst*.

As it runs, PEST produces many files. The number and type of files which it writes depends on its mode of operation, and on whether model runs are parallelised. Many of these files have the same filename base as the PEST control file. Thus, for example, if *case* is used to represent the filename base of the PEST control file, then the run record file recorded by PEST is named *case.rec* while the binary file used to store the Jacobian matrix is named *case.jco*.

A complete list of files written by PEST is provided in Appendix B of this document.

3.3 Modes of Operation

3.3.1 General

PEST has four modes of operation. On any particular run its mode is designated through the PESTMODE variable appearing in the "control data" section of the PEST control file. These modes are "estimation", "predictive analysis", "regularisation" and "pareto". A brief overview of each of these modes is provided shortly.

Regardless of its current mode, PEST's behavior is iterative. That is, a similar sequence of calculations is repeated many times. As explained by Doherty (2015), only one iteration would be required if a model was linear; repetition of iterations is required because sensitivities of model outputs to parameters change as the values of parameters change.

Each iteration begins with calculation of a Jacobian matrix. The Jacobian matrix has a column for each adjustable parameter (i.e. each parameter which is neither fixed nor tied to a parent parameter). It has a row for each observation and each item of prior information appearing in the PEST control file. Let J_{ij} denote the element of the Jacobian matrix that occupies its *i*'th row and *j*'th column. This element is the partial derivative of the *i*'th observation with respect to the *j*'th parameter. If a parameter is log-transformed during the inversion process (see later), then the partial derivative takes this into account.

Calculation of the Jacobian matrix requires at least as many model runs as there are adjustable parameters if elements of this matrix are filled by taking finite parameter differences. Hence this part of each iteration is normally the most numerically intensive part. However it is also the part of each iteration whose numerical burden is most easily lightened through parallelization of model runs, as each model run is independent of every other model run.

Once a Jacobian matrix has been calculated it can be used to calculate an improved set of parameters. However this calculation is normally undertaken using a number of different

values of the so-called Marquardt lambda. Different Marquardt lambdas are tested on a kind of "smart trial and error" basis. This strategy brings with it a number of advantages, these including the following:

- it accommodates nonlinear behaviour of a model's outputs with respect to its parameters;
- it constitutes a primitive regularisation device;
- it can provide some defense against corruption of finite-difference derivatives incurred by model numerical granularity.

This second part of each iteration is not as immediately parallelizable as the first part of each iteration, especially where parameters hit their bounds. In the latter case PEST actually re-formulates the inverse problem by temporarily freezing parameters at their bounds according to certain ordering rules; the advantages of this strategy are explained in section 5.4.2.7 of Doherty (2015). Nevertheless, with some compromises, PEST allows parallelisation of this part of each iteration to the degree requested by the user.

3.3.2 Estimation Mode

When the PESTMODE variable is set to "estimation" PEST employs no Tikhonov regularisation in solution of the inverse problem. Solution of the inverse problem can be based on the Gauss-Marquardt-Levenberg method; see section 5.4.2 of Doherty (2015). Alternatively, singular value decomposition or LSQR can be used to solve the inverse problem; both of these guarantee numerical stability even where an inverse problem is ill-posed; see section 6.2 of Doherty (2015). LSQR is similar to singular value decomposition in that it attempts to subdivide parameter space into orthogonal null and solution spaces; it then restricts solution of the inverse problem to the latter space. Where many parameters require estimation, its operation is much faster than that of singular value decomposition.

If prior information is featured in an "estimation mode" PEST control file, it is treated just like other observations (prior information can be considered as direct observations of parameter values, or of relationships between parameter values). In particular, weights applied to prior information equations are not altered as the inversion process progresses as is the case if prior information is employed as an agent for Tikhonov regularisation.

If the inverse problem is well-posed, and if neither singular value decomposition nor LSQR are used for solution of that problem, then PEST calculates a linear approximation to the posterior parameter covariance matrix during every iteration of the inversion process, and at the end of the inversion processes. This covariance matrix is proportional to the inverse of $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ where $\mathbf{J}$ is the Jacobian matrix and $\mathbf{Q}$ is the weight matrix; see section 5.3 of Doherty (2015). The posterior correlation coefficient matrix, as well as eigenvalues and eigenvectors of the posterior covariance matrix are also computed. If the inverse problem is nonunique then these are not calculated as the $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix cannot then be inverted. Instead PEST records the noninvertibility of this matrix to its run record file, and to an iteration-specific matrix file as the inversion process progresses. However if the inverse problem borders on nonuniqueness, then this can be gleaned from an inspection of the eigenvalues and eigenvectors of the posterior covariance matrix; ameliorative action can then be taken before repeating the inversion process. If the problem is well posed, parameter uncertainties and confidence limits (calculated under an assumption of model linearity) are recorded in the run record file upon completion of a PEST run.

If a covariance matrix is not recorded, this does not mean that post-calibration parameter

uncertainties cannot be quantified. This is easily accomplished using PEST utilities which were written for this purpose. See, in particular, the PREDUNC7 utility. PREDUNC7 computes a post-calibration parameter covariance matrix, regardless of the uniqueness or otherwise of the inverse problem. Other PEST utilities can compute quantities such as parameter identifiability, as well as uncertainties associated with predictions made by a calibrated or uncalibrated model.

3.3.3 Regularisation Mode

PEST's "regularisation" mode supports the use of Tikhonov regularisation in the inversion process; see section 6.3 of Doherty (2015) for concepts and theory. This, in turn, supports the use of highly parameterized inversion in place of parsimonious parameterisation as a regularization device, as an inverse problem does not need to be well-posed for model calibration to proceed. In fact, in most environmental modelling contexts, the model calibration process is better served if the innate heterogeneity of natural systems is embraced rather than eschewed. As Doherty (2015) describes, if regularization constraints are properly formulated, highly parameterized inversion yields a parameter field which is of minimum error variance, at the same time as this parameter field supports parameter and predictive uncertainty analysis that does not underestimate the true uncertainties of parameters, and of predictions of management interest.

When run in "regularisation" mode, PEST employs two objective functions rather than one. One of these is called the "measurement objective function" while the other is referred to as the "regularisation objective function". The lower is the former, the better do model-calculated quantities match their field-measured counterparts. The lower is the latter, the better do parameter values respect mathematical expressions of expert knowledge encapsulated in observations or prior information equations that are dedicated to this purpose.

A parameter covariance matrix is not recorded in either the run record file or in iteration-specific matrix files if PEST is run in "regularisation" mode. However posterior parameter and predictive uncertainties can be investigated using utilities such as PREDUNC7 and other members of the PREDUNC* suite, as well as the PREDVAR* and other utilities described in part II of this manual.

3.3.4 Predictive Analysis Mode

When PEST is run in "predictive analysis" mode it solves a constrained maximisation/minimisation problem in which a prediction of interest is maximised or minimised subject to the constraint that the objective function rises no higher than a user-specified level. Theory on which this mode of PEST's operation is based was derived by Cooley and Vecchia (1987) and Vecchia and Cooley (1987), and is described in detail in section 8.4 of Doherty (2015).

Use of PEST in "predictive analysis" mode should follow solution of a well posed, or only slightly ill-posed, inverse problem in which a model has been calibrated through reduction of an appropriate objective function to its minimum. Let us denote this minimised objective function as Φ_{min}. Let Φ_0 denote an objective function that is slightly higher than Φ_{min} above which the model is deemed to be "uncalibrated" at a certain level of confidence. Let *s* be a model output read by PEST that is actually a prediction, and is hence unused by the calibration process. The range of post-calibration uncertainty of this prediction can be determined through solution of two constrained optimisation problems in which the prediction is maximised, and then minimized, subject to the constraint that the objective

function is equal to Φ_0.

This means of assessing predictive uncertainty can work well where a model does not have too many parameters, and where it is numerically well-behaved so that finite-difference-based derivatives of model outputs with respect to parameters have a high degree of integrity. If these conditions are not met then other methods of predictive uncertainty assessment should be pursued; these can include linear methods, null space Monte Carlo, and PEST's "pareto" mode.

3.3.5 Pareto Mode

The name "Pareto" (after the Italian economist Vilfredo Pareto) is often given to methods which play one objective function against another. When run in "pareto" mode, PEST can play a measurement objective function against a regularisation objective function, or an objective function which respects both measurement data and expert knowledge against one that is lowered in accordance with proximity to a predictive value of management interest. In doing so, PEST explores the so-called "Pareto front" – a hypersurface in parameter space from which it is not possible to lower both objective functions simultaneously. By traversing this surface, and by presenting the outcomes of this traversal to a modeller, PEST allows the modeller to assess the cost in terms of one objective function to be paid for lowering another. This allows him/her to make qualitative assessments of the goodness of fit achieved with a calibration dataset, and whether that fit is "too good". In the predictive context it supports direct testing of hypotheses that predictions of unwanted environmental events will (or will not) occur if certain courses of management action are followed. See section 8.5 of Doherty (2015) for further details.

PEST can be used in "pareto" mode even where the number of adjustable parameters is large. Run-time burdens incurred through inclusion of a large number of parameters in Pareto analysis can be mitigated through model run parallelisation and/or through use of PEST's SVD-assist capabilities.

If desired, linear analysis of parameter and predictive uncertainty using utility programs described in part II of this manual can be undertaken after PEST is run in any of the modes discussed above using Jacobian matrices computed by PEST as it ran.

3.3.6 Another "Mode"

If started using the "/f" command line switch, PEST does not undertake parameter estimation at all. Instead, it carries out a series of model runs in order to calculate model outputs corresponding to different sets of parameters. In some ways this can be considered a different mode of behavior. See section 5.1.4 for details.

3.4 Parameter Adjustment

3.4.1 General

Many of the variables which govern the ways in which parameters are adjusted are featured in the "parameter data" section of the PEST control file. Additionally some variables which apply to all parameters, and/or to the inversion process in general, are featured in the "control data" section of the PEST control file. The roles of some of these variables are now discussed.

3.4.2 Parameter Transformation

PEST's task is to adjust parameters. However it can be asked, behind the scenes, to adjust the logarithms of some or all parameters instead of the parameters themselves. This often makes an inversion process more numerically stable, and faster, than it would otherwise be. It also removes large inequalities in sensitivities between parameters which may be an artefact of the units employed for their representation. Hence, unless parameters can become zero or negative, log-transformation of all parameters should be considered.

In the "parameter data" section of the PEST control file PEST requires that each parameter be designated as untransformed, log-transformed, fixed or tied; the latter two options are discussed below. If a parameter is log-transformed, any prior information pertaining to that parameter must pertain to the log (to base 10) of that parameter. Also, elements of the covariance, correlation coefficient and eigenvector matrices calculated by PEST associated with that parameter pertain to the log of the parameter rather than to the parameter itself. However PEST's estimates of the parameter's value refer to the actual parameter. The same applies to confidence intervals calculated for this value, if these are listed in the PEST run record file.

PEST learns whether a parameter is log-transformed through the parameter-specific variable named PARTRANS. You should never ask PEST to log transform a parameter whose initial value is zero or negative, or to log transform a parameter whose lower bound is zero or less. If you attempt to do this, PEST (or PESTCHEK) will inform you of your mistake.

More complex parameter transformations can be undertaken using the PAR2PAR parameter preprocessor; see part II of this manual.

3.4.3 Fixed and Tied Parameters

A parameter can be referenced in a template file yet take no part in the parameter estimation process. In this case it must be declared as "fixed" in the PEST control file so that its value does not vary from that assigned to it initially. This is accomplished by setting the PARTRANS variable associated with that parameter to "fixed" in the "parameter data" section of the PEST control file.

PEST allows one or more parameters to be tied (i.e. linked) to a "parent" parameter. PEST does not estimate a value for a tied parameter; rather it adjusts the parameter during the inversion process such that it maintains the same ratio with its parent parameter as that provided through the initial estimates of these parameters. Thus tied parameters "piggyback" on their parent parameters. Note that a parameter cannot be tied to a parameter which is either fixed, or tied to another parameter itself. Parameters are tied through setting PARTRANS to "tied". Parent parameters of tied parameters are identified in the second part of the "parameter data" section of the PEST control file, as is described in the next chapter.

3.4.4 Upper and Lower Parameter Bounds

Upper and lower bounds, defining the maximum and minimum values which a parameter is allowed to assume during the inversion process, must be supplied for all parameters. These are provided through the PARLBND and PARUBND variables in the "parameter data" section of the PEST control file.

It is important that upper and lower parameter bounds be chosen wisely. For many models parameters can lie only within certain well-defined domains determined by the theory on which the model is based. In such cases model-generated floating-point errors may result if

PEST is not prevented from adjusting a parameter to a value outside its allowed domain. For example if, at some stage during a model run, the logarithm or square root of a particular parameter is taken, then that parameter must be prevented from ever becoming negative (or zero if the model takes the log of the parameter). If the reciprocal is taken of a parameter, the parameter must never be zero.

The algorithm used by PEST to enforce parameter bounds contributes to the numerical stability of the inversion process, at the same time as it allows PEST to find the lowest objective function within that part of parameter space in which it is allowed to roam. See section 5.4.2.7 of Doherty (2015) for details.

3.4.5 Optional Alternative Bounds Accommodation

If a model takes a long time to run, then any measures that can be taken to reduce the number of runs required for estimation of its parameters makes that model more useable with PEST. Parameter "bounds-sticking" functionality is one such measure.

When one or more parameters are at their bounds, the procedure by which the parameter upgrade vector is calculated is modified to accommodate this situation. If, for one such parameter, the parameter upgrade vector is such that the parameter is directed away from its bound back into "allowed parameter space", then that parameter is free to move as it normally would. However parameters for which the upgrade vector points outside of allowed parameter space (i.e. above or below the upper or lower bound at which the parameter currently resides) are sequentially frozen; the sequence in which these parameters are frozen is important; see Doherty (2015). On each occasion that a parameter is frozen, the parameter upgrade vector is re-calculated with currently frozen parameters omitted from the parameter upgrade calculation. At the beginning of each new iteration, all frozen parameters are freed. The whole process is then repeated during the next iteration of the inversion process.

Sometimes a parameter can move to its bound, and then back again into allowed parameter space, during the course of a single inversion process. However if a parameter is at its bound for more than a few iterations, it is more likely than not that the parameter is there to stay. However PEST will continue to calculate derivatives with respect to this parameter in order to attempt an upgrade calculation for it, even though there is a diminishing likelihood that the parameter will ever move from its bounds. Thus one model run per iteration (more if PEST is engaged in higher order finite-difference derivatives calculation) is wasted. Where more than one parameter is at its bound the number of wasted model runs rises in proportion to the number of such parameters.

Through use of the IBOUNDSTICK control variable residing in the "control data" section of the PEST control file, this wastage of model runs can be prevented. PEST can be instructed to permanently "glue" a parameter to its upper or lower bound if that parameter has been residing there for a user-specified number of iterations. Once a parameter is "glued" to its bound it will never move again, for PEST will no longer include this parameter in its upgrade vector calculations. Nor will it calculate derivatives with respect to this parameter, thus reducing the number of model runs required per iteration.

Despite the potential for model run economy that it offers, bounds-sticking functionality should be used with caution.

The UPVECBEND variable (which follows the IBOUNDSTICK variable in the "control data" section of the PEST control file) activates "upgrade parameter vector bending" functionality. Experience has demonstrated that, while this may result in a slight reduction in

the number of model runs required for completion of the parameter estimation process in some situations, it can seriously degrade PEST's performance in other situations. Hence it should be used only with extreme caution.

As is described by Doherty (2015), and also below, the length of the parameter upgrade vector is limited during any one PEST iteration by the action of factor, relative and/or absolute parameter change limits. Imposition of limits prevents parameter changes from exceeding by too great a margin the range of the linearity assumption upon which their calculation is based. In doing so, it brings stability to the inversion process.

If PEST calculates that a particular parameter must incur a factor, relative or absolute change which is greater than that permitted by pertinent user-supplied limits, then the changes incurred by *all* parameters are reduced so that the change incurred by the maximally-changed parameter does not exceed its limit; that is, the length of the upgrade vector is reduced to respect this limit while its direction remains unchanged. As it is often the most insensitive parameters for which the largest change is calculated (especially if regularisation is omitted from a PEST control file), it is on behalf of these parameters that the parameter upgrade vector may thus be shortened. If this is the case, then changes incurred by the more important, sensitive, parameters can thus be curtailed by this shortening of the parameter upgrade vector. This may slow the inversion process.

The UPVECBEND variable can instruct PEST not to shorten the parameter upgrade vector in its observance of parameter change limits. Instead it will "bend" this vector so that, while individual parameters are forced to respect their change limits, other parameters can move by the amount dictated by the parameter upgrade vector. Hence changes calculated for sensitive parameters are not restricted by the change limits imposed on more insensitive parameters.

3.4.6 Scale and Offset

A scale and offset (variables SCALE and OFFSET in the "parameter data" section of the PEST control file) must be supplied for all parameters. Before writing a parameter value to a model input file, PEST multiplies the parameter's value by the scale and adds the offset.

The SCALE and OFFSET variables can be very convenient in some situations. For example, in a particular model, a certain parameter may have a non-zero base level; you may wish to re-define the parameter as the actual parameter minus this base level. Elevation may be such a parameter. If a reference elevation is subtracted from the true, model-required elevation, the result may be thickness; this may be a more "natural" parameter for PEST to estimate than elevation. In particular, it may make more sense to express a derivative increment (see below) as a fraction of thickness than as a fraction of elevation, to which an arbitrary datum has been added. Also, the inversion process may be better behaved if the thickness parameter is log-transformed; in contrast, it would be surprising if log-transformation of elevation improved inversion performance due to the arbitrary datum with respect to which an elevation must be expressed. PEST can thus estimate thickness, converting this thickness to elevation every time it writes a model input file by adding the reference elevation stored as the parameter's OFFSET.

The SCALE variable is equally useful. A model's parameter may be such that it can only be ascribed negative values; such a parameter cannot be log-transformed. However if a new parameter is defined as the negative of the model-required parameter, PEST can estimate this new parameter, log-transforming it if necessary to enhance inversion efficiency. Just before it writes the parameter to a model input file, PEST multiplies it by its SCALE (-1 in this case)

so that the model receives the parameter it expects.

If you do not wish a parameter to be scaled and offset, provide its SCALE as 1 and its OFFSET as zero.

It should be stressed that PEST is oblivious to a parameter's SCALE and OFFSET until the moment it writes its value to a model input file. It is at this point (and only this point) that it first multiplies by the SCALE and then adds the OFFSET; the SCALE and OFFSET take no other part in the parameter estimation process. Note also that fixed and tied parameters must each be supplied with a SCALE and OFFSET, just like their adjustable (log-transformed and untransformed) counterparts.

3.4.7 Global Parameter Scaling

Ideally, especially if an inverse problem is ill-posed, parameters should be subjected to a Kahunen-Loève transformation prior to estimation such that they show no prior statistical correlation and so that their prior standard deviations are all the same; they should then be back-transformed after estimation. See section 6.2.4 of Doherty (2015) for details. In real-world inversion practice it is rarely possible to do this. However some of the benefits of this approach may be gained if parameters are internally normalized with respect to ranges defined by their bounds. This can be accomplished through appropriate setting of the BOUNDSCALE variable in the "control data" section of the PEST control file.

3.4.8 Parameter Change Limits

No parameter can be adjusted by PEST above its upper bound or below its lower bound. However, as has already been discussed, PEST places a further limit on the amount by which a parameter is permitted to change in any one iteration of the inversion process.

Unless the model under PEST's control exhibits outrageously nonlinear behaviour, an updated parameter set calculated using one of the inverse problem solution strategies discussed by Doherty (2015) will result in a lowering of the objective function. (The objective function is defined in accordance with PEST's current mode of operation.) However if a model is highly nonlinear, the parameter upgrade vector may "overshoot" the objective function minimum, and the new value of the objective function may actually be worse than the old one. This is because the equations employed for calculation of the upgrade vector are all based on a local quasi-linearity assumption which may not extend as far into parameter space from current parameter estimates as the magnitude of the upgrade vector itself.

To obviate the possibility of overshoot, it is good practice to place a reasonable limit on the maximum change that any adjustable parameter is allowed to undergo in any one iteration. Such limits may be of three types, namely "relative", "factor" and "absolute". You must inform PEST, through the parameter-specific variable PARCHGLIM (residing in the "parameter data" section of the PEST control file) which type of change limit applies to each adjustable parameter. Variables RELPARMAX and FACPARMAX provide the maximum allowed relative and factor changes for all relative-limited and factor-limited parameters respectively. Meanwhile ABSPARMAX(*N*) (where *N* can range from 1 to 10) provides the absolute change limit for parameters assigned to absolute change group *N*. These variables appear in the "control data" section of the PEST control file.

Let f represent the user-defined maximum allowed parameter factor change for factor-limited parameters (i.e. FACPARMAX); f must be greater than unity. Then if b_0 is the value of a

particular factor-limited parameter at the beginning of a PEST iteration, the value b of this same parameter at the beginning of the next iteration will lie between the limits

$$b_0/f \le b \le fb_0 \qquad (3.4.1a)$$

if b_0 is positive, and

$$fb_0 \le b \le b_0/f \qquad (3.4.1b)$$

if b_0 is negative. *Note that if a parameter is subject to factor-limited changes, it can never change sign*.

Let r represent the user-defined maximum allowed relative parameter change for all relative-limited parameters (i.e. RELPARMAX); r can be any positive number. Then if b_0 is the value of a particular relative-limited parameter at the beginning of a PEST iteration, its value b at the beginning of the next iteration will be such that

$$\frac{|b-b_0|}{|b_0|} \le r \qquad (3.4.2)$$

In this case, unless r is less than or equal to unity, a parameter can, indeed, change sign. However there may be a danger in using a relative limit for some types of parameters in that if r is 1 or greater, b may fall to a minute fraction of b_0 (or even to zero), without transgressing the parameter change limit. For some types of parameters in some models this will be fine; in other cases a parameter factor change of this magnitude may significantly transgress model linearity limits.

Suppose that, for a particular parameter, the parameter-specific variable PARCHGLIM is specified as "absparmax(N)" where N can range from 1 to 10. Suppose that ABSPARMAX(N) in the "control data" section of the PEST control file is assigned a value of a. Then, in any one iteration, PEST will limit changes to the value of this parameter such that

$$|b - b_0| < a \qquad (3.4.3)$$

In implementing the conditions set by equations 3.4.1 to 3.4.3, PEST limits the magnitude of the parameter upgrade vector such that none of these equations is violated for any parameter to which the pertinent equation applies. Naturally, if only one type of parameter change limit is featured in a current PEST run (for example if parameters are all factor-limited or are all relative-limited) only the pertinent one of these limits is invoked.

If, in the course of the inversion process, PEST assigns to a parameter a value which is very small relative to its initial value, then either of equations 3.4.1 or 3.4.2 may place an undue restriction on subsequent parameter adjustments. Thus if b_0 for one parameter is very small, the changes to all parameters may be set intolerably small so that equation 3.4.1 or equation 3.4.2 is obeyed for this one parameter. To circumvent this problem, PEST provides another variable, FACORIG (in the "control data" section of the PEST control file) which allows the user to limit the effect that an unduly low-valued parameter value can have in this regard. If the absolute value of a parameter is less than FACORIG times its initial absolute value and PEST wishes to adjust that parameter such that its absolute value will increase, then FACORIG times its initial value is substituted into equation 3.4.1 and the denominator of equation 3.4.2 for the parameter's current value b_0. A suitable value for FACORIG varies from case to case, though 0.001 is often appropriate. Note, however, that FACORIG is not used to adjust change limits for log-transformed parameters or for parameters that are subject

to an absolute change limit.

Problems associated with imposing changes on relative- or factor-limited parameters when their values become very low can be prevented by declaring affected parameters as absolute-limited. Alternatively, such parameters can be provided with a suitable OFFSET value, accompanied by appropriately offset lower/upper bounds that prevent them from being assigned such troublesome values.

3.4.9 Damping of Parameter Changes

Regardless of PEST's mode of operation, parameter over-adjustment, and any oscillatory behaviour that results from this, is mitigated by its "dampening" of parameter changes. The method used by PEST is based on a technique described by Cooley (1983) and used by Hill (1992). To see how it works, suppose that a parameter upgrade vector **u** has just been determined. Suppose, further, that this upgrade vector causes no parameter values to exceed their bounds, and that all parameter changes are within factor, relative and absolute limits.

For relative-limited parameters, let the parameter undergoing the proposed relative change of greatest magnitude be parameter *i*; let its proposed relative change be p_i. For absolute-limited parameters, let the parameter undergoing the proposed absolute change of greatest magnitude be parameter *m*; let its proposed absolute change be a_i. For factor-limited parameters which are not log-transformed, define q_j for parameter *j* as

$$\begin{aligned} q_j &= u_j / (f b_j - b_j) && \text{if } u_j \text{ and } b_j \text{ have the same sign, and} \\ q_j &= u_j / (b_j - b_j / f) && \text{if } u_j \text{ and } b_j \text{ have the opposite sign} \end{aligned} \qquad (3.4.4)$$

where b_j is the current value of the *j*'th parameter and *f* is the maximum allowed factor change for all factor-limited parameters. Let the parameter for which the absolute value of *q* is greatest be parameter *l*, and let *q* for this parameter be q_l. Finally, let the log-transformed parameter for which the absolute value of **u** is greatest be parameter *k*, and let the element of **u** pertaining to this parameter be u_k. Let i_0, m_0, l_0, k_0, p_{0i}, a_{0i}, q_{0l} and u_{0k} define these same quantities for the previous iteration except that, for the previous iteration, they are defined in terms of actual parameter changes rather than proposed ones. Now define s_1, s_2, s_3 and s_4 such that

$$\begin{aligned} s_1 &= p_i / p_{0i} && \text{if } i = i_0; \\ s_1 &= 0 && \text{otherwise,} \end{aligned} \qquad (3.4.5a)$$

$$\begin{aligned} s_2 &= a_m / a_{0m} && \text{if } m = m_0; \\ s_2 &= 0 && \text{otherwise,} \end{aligned} \qquad (3.4.5b)$$

$$\begin{aligned} s_3 &= q_l / q_{0l} && \text{if } l = l_0; \\ s_3 &= 0 && \text{otherwise, and} \end{aligned} \qquad (3.4.5c)$$

$$\begin{aligned} s_4 &= u_k / u_{0k} && \text{if } k = k_0; \\ s_4 &= 0 && \text{otherwise.} \end{aligned} \qquad (3.4.5d)$$

Let *s* be the minimum of s_1, s_2, s_3 and s_4 and define ρ as:

$$\rho = (3 + s)/(3 + |s|) \quad \text{if } s \geq -1 \qquad (3.4.6a)$$

$$\rho = 1/(2|s|) \quad \text{otherwise.} \qquad (3.4.6b)$$

Then oscillatory behaviour of the inversion process is mitigated by defining a new parameter

upgrade vector **v** by

$$\mathbf{v} = \rho\mathbf{u} \qquad (3.4.7)$$

3.5 The Calculation of Derivatives

3.5.1 General

The ability to calculate partial derivatives of all observations with respect to all adjustable parameters is fundamental to implementation of the inversion methodologies supported by PEST. These derivatives are stored as the elements of the Jacobian matrix. Because PEST is independent of any model of which it takes control, it cannot calculate these derivatives using formulae specific to the model. Hence it must evaluate the derivatives itself using model outputs calculated on the basis of incrementally varied parameter values. Note, however, that there may be occasions where a model can calculate derivatives of its outputs with respect to its adjustable parameters itself. If this is the case, PEST can make direct use of these derivatives if they are provided to it in the correct format. This is further discussed later in this manual.

Accuracy in derivatives calculation is fundamental to good PEST performance. In early versions of PEST, failure to minimise an objective function may sometimes have been attributable to ill-posedness of an inverse problem. The Gauss-Marquardt-Levenberg method relies on inversion of the so-called "normal" matrix (enhanced by the addition of the Marquardt lambda to its diagonals) to calculate an updated parameter set. Where an inversion problem is ill-posed this matrix is singular and hence cannot be inverted. Hence progress of the parameter estimation process becomes impossible.

These days things are very different. Use of singular value decomposition or LSQR to solve the inverse problem guarantees unconditional numerical stability of the solution process, whether the problem is well-posed or not. The addition of Tikhonov regularisation guarantees achievement of a parameter set of minimum error variance in solution of that problem. The only thing that can prevent PEST from solving an inverse problem is lack of integrity of partial derivatives contained in the Jacobian matrix. This can be caused by the presence of round-off errors in taking differences of small quantities, as is required for computation of partial derivatives using finite differences. It can also be an outcome of poor model numerical performance whereby changes to model outputs following incremental changes in parameters are caused not only by alterations to parameter values, but also by problematic convergence of the model's solver, or by the presence of artificial thresholds and discontinuities in its simulation algorithm. Fortunately, as the present section shows, it may still be possible to use PEST with a model even it is falls victim to numerical idiosyncrasies which compromise differentiability of model outputs with respect to adjustable parameters. However if its numerical behaviour is too degraded, then PEST's performance may suffer to the point where it cannot be used with that model. In this case a global optimiser must be used to calibrate the model; access is thereby lost to the advanced regularisation features provided by PEST.

The PEST input variables which govern finite-difference derivatives calculation are assigned to parameter "groups". In the PEST control file, each parameter must be assigned to a group. The assignment of derivative-pertinent variables to groups, rather than to individual parameters, introduces savings in memory and complexity of the PEST input dataset. Furthermore, in many instances, parameters naturally fall into one or more categories. For example, if the domain of a two- or three-dimensional spatial model is parameterized using

pilot points, pilot-point parameters which describe system properties of the same type would normally be assigned to the same group. However, if you wish to treat each parameter differently as far as derivatives calculation is concerned, this can be achieved by assigning each parameter to a group of its own.

3.5.2 Forward, Central and Five-Point Differences

The simplest way to calculate derivatives is through the method of forward differences. To calculate derivatives in this manner, PEST varies each parameter in turn by adding an increment to its current value (unless the current parameter value is at its upper bound, in which case PEST subtracts the increment). Then PEST runs the model, reads the altered, model-generated observations and then approximates the derivative of each observation with respect to the incrementally-varied parameter as the observation increment divided by the parameter value increment. (For log-transformed parameters this quotient is then multiplied by the current parameter value times the natural log of 10.) Hence if derivatives with respect to all parameters are calculated by the method of forward differences, the filling of the Jacobian matrix requires that a number of model runs be carried out equal to the number of adjustable parameters; as the Jacobian matrix must be re-calculated during every PEST iteration, each iteration requires at least as many model runs as there are adjustable parameters (plus at least another one to test parameter upgrades). The calculation of derivatives is by far the most time-consuming part of the inversion process.

If the parameter increment is properly chosen (see below), this method can work well. However it is often found that as the objective function minimum is approached, attainment of this minimum requires that parameters be calculated with greater accuracy than that afforded by the method of forward differences. Thus PEST also allows for derivatives to be calculated using three parameter values and corresponding observation values rather than two, as are used in the method of forward differences. Experience shows that derivatives calculated on this basis are accurate enough for most occasions provided, once again, that parameter increments are chosen wisely. As three-point derivative calculations are normally carried out by first adding an increment to a current parameter value and then subtracting an increment, the method is referred to herein as the "central" method of derivatives calculation. Note that if a parameter value is at its upper bound, the parameter increment is subtracted once and then twice, the model being run each time; if it is at its lower bound the increment is added once and then twice.

PEST uses one of three methods to calculate central derivatives. In the first or "outside points" method, the two outer parameter values (i.e. that for which an increment has been added and that for which an increment has been subtracted) are used in the same finite-difference type of calculation as is used in the forward difference method. This method yields more accurate derivative values than the forward difference method because the (unused) current parameter value is at the centre of the finite-difference interval (except where the parameter is at its upper or lower bound). The second method is to define a parabola through the three parameter-observation pairs and to calculate the derivative of this parabola with respect to the incrementally-varied parameter at the current value of that parameter. This method, referred to as the "parabolic" method, can yield very accurate derivatives if model-calculated observation values can be read from the model output file with sufficient precision. The third method is to use the least-squares principal to define a straight line of best fit through the three parameter-observation pairs and to take the derivative as the slope of this line. This method may work best where model-calculated observations cannot be read from model output files with great precision, because of either deficiencies in the model's

numerical solution method, or because the model writes numbers to its output files using a limited number of significant figures.

If the central method of derivatives calculation is used for all parameters, each iteration of the inversion process requires that at least twice as many model runs be carried out as there are adjustable parameters. If the central method is used for some parameters and the forward method is used for others, the number of model runs will lie somewhere between the number of adjustable parameters and twice the number of adjustable parameters.

PEST provides the further option of using a five-point finite-difference stencil for calculation of derivatives for some or all parameters. This requires that four model runs be carried out per parameter. Parameter values used in this procedure are symmetrically disposed with respect to the current parameter value; a fifth model run is not required to compute this value as it is already known.

The situation is depicted in figure 3.1.

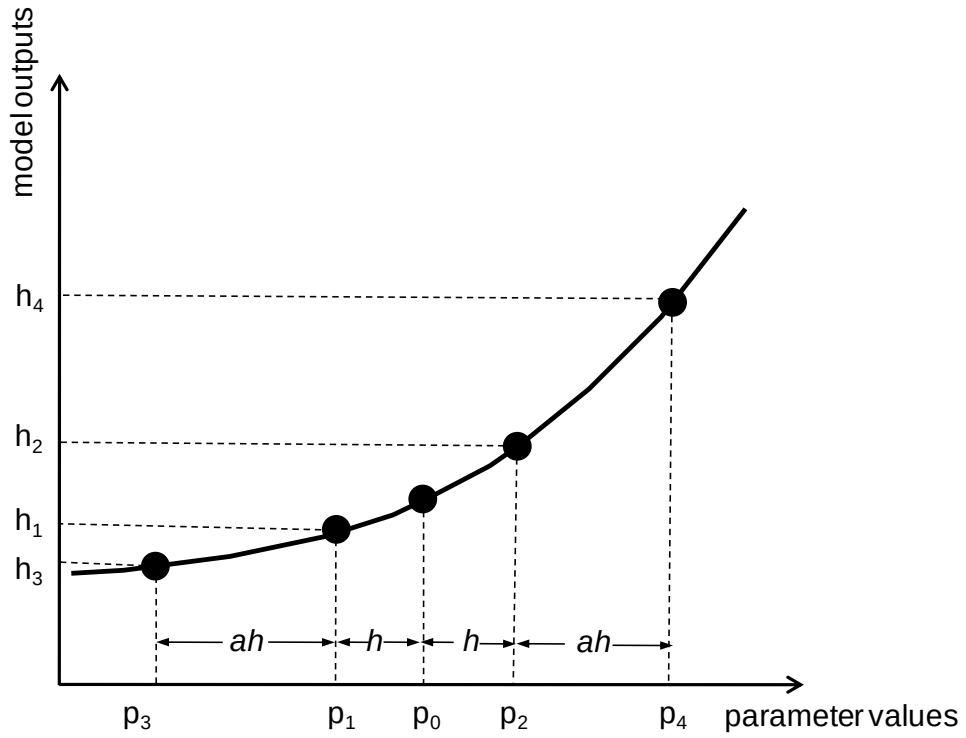


Figure 3.1 Five point derivatives stencil.

The derivative of model outputs with respect to parameter values at a parameter value of p_0 is given by

$$dh/dp \approx \alpha(h_2 - h_1) + \beta(h_4 - h_3) \qquad (3.5.1)$$

Nearing (2001) discusses two options for choice of α and β. Maximum precision (elimination of error terms before the fourth derivative) is attained through choice of α and β such that

$$\alpha = a^2/2h(a^2 - 1) \qquad (3.5.2)$$

$$\beta = -1/2ha(a^2 - 1) \qquad (3.5.3)$$

On the other hand, a minimum error variance estimate of the derivative at p_0 can be obtained with α and β selected such that

$$\alpha = 1/2h(a^2 + 1) \quad (3.5.4)$$

$$\beta = a/2h(a^2 + 1) \quad (3.5.5)$$

The latter choice is far better where model outputs are contaminated by numerical noise that may arise from model solution convergence difficulties and other factors. It would normally be to address this problem that a modeller would even consider undertaking four runs per parameter per iteration.

3.5.3 Parameter Increments for Two and Three-Point Derivatives

Because of the importance of reliable finite-difference derivatives calculation, PEST provides considerable flexibility in the way parameter increments are chosen. Mathematically, a parameter increment should be as small as possible so that the finite-difference method (or one of its higher order variants) provides a good approximation to the derivative in a theoretical sense (remember that the derivative is defined as the limit of the finite difference as the increment approaches zero). However, if the increment is made too small, accuracy of derivatives calculation suffers because of the effect of round-off errors as two, possibly large, numbers are subtracted to yield a much smaller number. In most cases intuition and experience, backed up by trial and error, will be your best guide in reconciling these conflicting demands on increment size.

As stated above, PEST variables which control derivatives calculation are assigned to parameter groups, rather than to parameters themselves. These variables reside in the "parameter groups" section of the PEST control file.

Three PEST input variables, namely INCTYP, DERINC and DERINCLB are of primary relevance to the setting of parameter increments. INCTYP determines the type of increment to use, for which there are three options, namely "absolute", "relative" and "rel_to_max". If the increment type for a parameter group is "absolute", the increment used for all parameters in the group is supplied as the input variable DERINC; this increment is added (and subtracted for central derivatives calculation) directly to a particular group member when calculating derivatives with respect to that parameter. However if the increment type is "relative", DERINC is multiplied by the current absolute value of a parameter in order to determine the increment for that parameter. In this way the parameter increment is adjusted upwards and downwards as the parameter itself is adjusted upwards and downwards; this may have the effect of maintaining significance in the difference between model outcomes calculated on the basis of the incrementally varied parameter. If the increment type for a group is "rel_to_max", the increment for all members of that group is calculated as DERINC times the absolute value of the group member of currently greatest absolute value. This can be a useful means by which to calculate increments for parameters whose values can vary widely, including down to zero. The "relative" aspect of the "rel_to_max" option may maintain model outcome difference significance as described above; however, because the increment is calculated as a fraction of the maximum absolute value occurring within a group, rather than as a fraction of the value of each parameter, an individual parameter can attain near-zero values without its increment simultaneously dropping to zero.

A further measure to protect against the occurrence of near-zero increments for "relative" and "rel_to_max" increment types is provided through the variable DERINCLB. This variable contains a standby absolute increment which can be used in place of the "relative" or "rel_to_max" increment if the increment calculated for a particular parameter using either of these latter methods falls below the absolute increment value contained in DERINCLB.

If a parameter is log-transformed then it is wise that its increment be calculated using the "relative" method, though PEST does not insist on this.

As PEST reads the data contained in its control file, it will object if a parameter increment (either read directly as "absolute" or calculated from initial parameter values as "relative" or "rel_to_max") exceeds the range of values allowed for that parameter (as defined by the parameter's upper and lower bounds) divided by 3.2, as the increment is then too large compared with the width of the parameter domain. However should this eventuality arise later in the course of the estimation process (as may happen if certain parameter values grow large and increments calculated from them as "relative" or "rel_to_max" then exceed the parameter domain width divided by 3.2) PEST will automatically adjust the increment so that parameter limits are not transgressed as the increment is added and/or subtracted from the current parameter value for the calculation of derivatives.

When choosing an increment for a parameter, care must be taken to ensure that the parameter can be written to the model input file with sufficient precision to distinguish an incremented parameter value from one which has not been incremented. Thus, for example, if a model input file template is such that a particular parameter value must be written to a space which is four characters wide, and if the increment type for that parameter is "absolute" and the increment value is 0.0001 while the current parameter value is .01, it will not be possible to discriminate between the parameter with and without its increment added. To rectify this situation, you should either increase the parameter field width in the template file (making sure that the model can still read the parameter) or increase the increment to a value whereby the incremented parameter is distinguishable from the non-incremented parameter in a field width of only four characters.

It will be recalled that PEST writes a parameter value to a model input file with the maximum possible precision, given the parameter field width provided in the pertinent template file. Also, for the purposes of derivatives calculation, PEST adjusts a parameter increment to be exactly equal to the difference between a current parameter value and the incremented value of that parameter as represented (possibly with limited precision) in the model input file, as read by the model.

3.5.4 Settings for Higher Order Derivatives

The parameter group input variable FORCEN determines whether derivatives for the parameters of a particular group are calculated using the forward-difference method, the central-difference method, the five-point method, or whether the method should change as the inversion process progresses. FORCEN can be designated as "always_2", "always_3", "switch" or "switch_5". If it is supplied as "always_2", derivatives calculation is through forward differences for all parameters within the group for the duration of the inversion process; if it is "always_3", central (i.e. three-point) derivatives will be used for the entirety of the inversion process; similarly if it is "always_5" then a five-point derivatives stencil is employed at all times for members of that parameter group. However if FORCEN is supplied as "switch", PEST will commence the inversion process using forward differences for all members of the group, and switch to using central differences on the first occasion that the relative reduction in the objective function between iterations is less than the value of the variable PHIREDSWH (which resides in the "control data" section of the PEST control file). Alternatively if it is set to "switch_5" the switch is made from forward differences to five-point derivatives under the same conditions. PEST provides no option for switching from three-point to five-point derivatives.

Operation of the PHIREDSWH variable can be over-ridden by the NOPTSWITCH variable which, like PHIREDSWH, also resides in the "control data" section of the PEST control file. If this is set to *N*, the switch from forward to central or five-point derivatives will not take place until iteration *N*. This prevents premature switching to a more model run intensive derivatives calculation methodology where early progress in lowering of the objective function may have been hampered by the imposition of parameter change limits, or through shortening of the parameter upgrade vector as parameters encounter their bounds.

Two parameter group variables pertain specifically to the calculation of derivatives using higher order methods. These are DERINCMUL and DERMTHD. If FORCEN is set to "always_3" or "switch", DERMTHD must be set to one of "outside_pts", "parabolic" or "best_fit"; this determines the method of central derivatives calculation to be used by PEST, the three options having already been discussed. If FORCEN is set to "always_5" or "switch_5" then DERMTHD must be supplied as either "minvar" to activate the minimum error variance alternative of the five-point methodology (as is recommended), or to "maxprec" to activate the maximum precision alternative.

The variable DERINCMUL is the parameter increment multiplier; this is the value by which DERINC is multiplied when it is used to evaluate increments for any of the three central derivatives methods. Sometimes it is useful to employ larger increments for central derivatives calculation than for forward derivatives calculation, especially where the model output dependence on parameter values is "bumpy" (see the next section). However if the increment is raised too high, derivative precision must ultimately fall. Note that through DERINCMUL you can also reduce the increment used for central derivatives calculation if you wish.

DERINCMUL plays a similar role in implementation of five-point derivatives as that which it plays in implementation of central derivatives. In figure 3.1 the distance *2h* is equal to the derivative increment calculated using DERINC (or DERINCLB as appropriate) multiplied by DERINCMUL. The factor *a* depicted in that figure is automatically set to 2.0 by PEST; hence the increments p_3 to p_1, p_1 to p_2 and p_2 to p_4 are always identical.

For increments calculated using the "relative" and "rel_to_max" methods, the variable DERINCLB has the same role in central and five-points derivatives calculation as it does in forward derivatives calculation, namely to place a lower limit on the absolute increment value. Note, however, that DERINCLB is not multiplied by DERINCMUL when derivatives are calculated using the central or five-point method.

3.5.5 How to Obtain Derivatives You Can Trust

Reliability of derivatives calculation can suffer if the model which you are trying to calibrate does not write its outcomes to its output file(s) using many significant figures. *If you have any control over the precision with which a model writes its output data, you should request that the maximum possible precision of representation be used*. Although PEST will happily attempt calibration on the basis of limited-precision model outputs, its ability to find an objective function minimum decreases as the precision of model-generated observations decreases. Furthermore, the greater the number of parameters which you are simultaneously trying to estimate, the greater will be the deleterious effects of limited model output precision on the inversion process.

If a model is comprised of multiple sub-model executables run by PEST through a batch file, then you should also ensure that numbers are transferred between these various sub-models

with maximum precision. Thus every sub-model comprising the composite model should record numbers to those of its output files which are read by other sub-models with full numerical precision.

Many models calculate their outcomes using one or a combination of numerical approximations to differential equations, for example the finite-difference method, finite-element method, boundary element method, etc. Problems which are continuous in space and/or time are approximated by discrete representations of the same problem in order that the partial differential equation(s) describing the original problem can be cast as a matrix equation of high order. The matrix equation is often solved by an iterative technique (for example preconditioned conjugate gradient, alternating direction implicit, etc.) in which the solution vector is successively approximated, the approximation being fine-tuned until it is judged that "convergence" has been attained. Most such iterative matrix solution schemes deem the solution to be acceptable when no element of the solution vector varies by more than a user-specified tolerance between successive iterations. If this threshold is set too large, model precision is reduced and finite-difference derivatives calculated by PEST suffer accordingly. If it is set too small, solution convergence may not be attainable; in any case, the smaller it is set, the greater will be the model run time.

Sometimes adaptive time-stepping schemes employed by highly non-linear models can contribute to numerical granularity of its outputs. Thus, for example, an incremental variation in the value of a particular parameter may lead to adoption of a different time-stepping strategy by a model from that which it employed for the non-incremented parameter. Differences in model outputs at a particular simulation time may then show some slight dependence on the solution trajectory, and therefore may not be entirely dependent on the difference in parameter values.

If numerical models of these types are to be used with PEST, *it is essential that any variables governing the numerical solution procedure be set in favour of precision over execution speed*. Although the model run-time may be much greater as a result, it would be false economy to give computation speed precedence over output precision. Accurate derivatives calculation depends on accurate calculation of model outcomes. If PEST is trying to estimate model parameters on the basis of imprecise model-generated observations, derivatives calculation will suffer, and with it PEST's chances of finding the optimal parameter set. Even if PEST is still able to find the optimal parameter set (which it often will), it may require more iterations to do so, resulting in a greater overall number of model runs, removing any advantages gained in reducing the time required for a single model run.

Even after you have instructed the model to write to its output file(s) with as much precision as possible, and you have adjusted the model's solution settings for greatest precision, model-generated observations may still be "granular" because of the nature of the simulator's algorithm. In this case it may be wise to set parameter increments larger than you normally would. If a parameter increment is set too small PEST may calculate a local, erroneous "bump" derivative rather than a derivative that reflects an observation's true dependence on a parameter's value. While use of a large increment incurs penalties due to poor representation of the derivative by a finite difference (especially for highly nonlinear models), this can be mitigated by the use of one of the three-point methods of derivatives calculation offered by PEST, particularly the parabolic and best-fit options. Due to its second order representation of the observation-parameter relationship, the parabolic method can generate reliable derivatives even for large parameter increments. However, if model outcomes are really bumpy, the best-fit method may be more accurate. Trial and error will determine the best

method for the occasion. As an extreme measure the minimum variance option of the five-point derivatives scheme can be used as a basis for finite-difference derivatives calculation.

The "split slope analysis" methodology provided by PEST offers yet another option for mitigating the effects of poor model numerical performance on PEST performance. This, and a means by which a better understanding of the numerical situation with which PEST is presented, are now described.

3.5.6 Looking at Model Outputs under the Magnifying Glass

Bad derivatives express themselves in various ways during a PEST run. More often than not however, a sudden refusal by PEST to lower the objective function any further, possibly accompanied by a "bouncing around" of the objective function as different parameter upgrades are tested using different Marquardt lambdas, is a sign of corrupted derivatives.

Included in the PEST suite are a number of utility programs which allow you to explore whether use of PEST with a particular model is being compromised by bad numerical performance of that model. See, for example, the JACTEST, POSTJACTEST and MULJCOSEN utilities described in part II of this manual. The JACTEST utility is particularly helpful in this regard. It undertakes a number of model runs with the value of one parameter varied a little each time by the same amount that it is varied for the purpose of finite-difference derivatives calculation. The values of one or many model outputs can then be plotted against the values taken by the parameter. Ideally, the thus-plotted curve should be smooth. (It can be straight or curved, but it should not be jagged.)

For many models, analysis using JACTEST reveals that contamination of derivatives by numerical noise is not necessarily purely random. The upper part of figure 3.2 illustrates JACTEST outcomes in which the value of a particular model output is plotted against the value of a parameter. This illustrates a situation wherein derivatives are contaminated by purely random numerical noise. In contrast, the lower part of the figure presents the more common situation, in which a plot of model outputs against incremented parameter value reveals segments in which derivatives do indeed have integrity. It is only in the vicinity of points (a) and (b) in this figure that derivatives are not useable; elsewhere they are indeed fit for use in the parameter estimation process. Experience suggests that such aberrant slopes often exist only between particular neighbouring pairs of points rather than being spread out over many points.

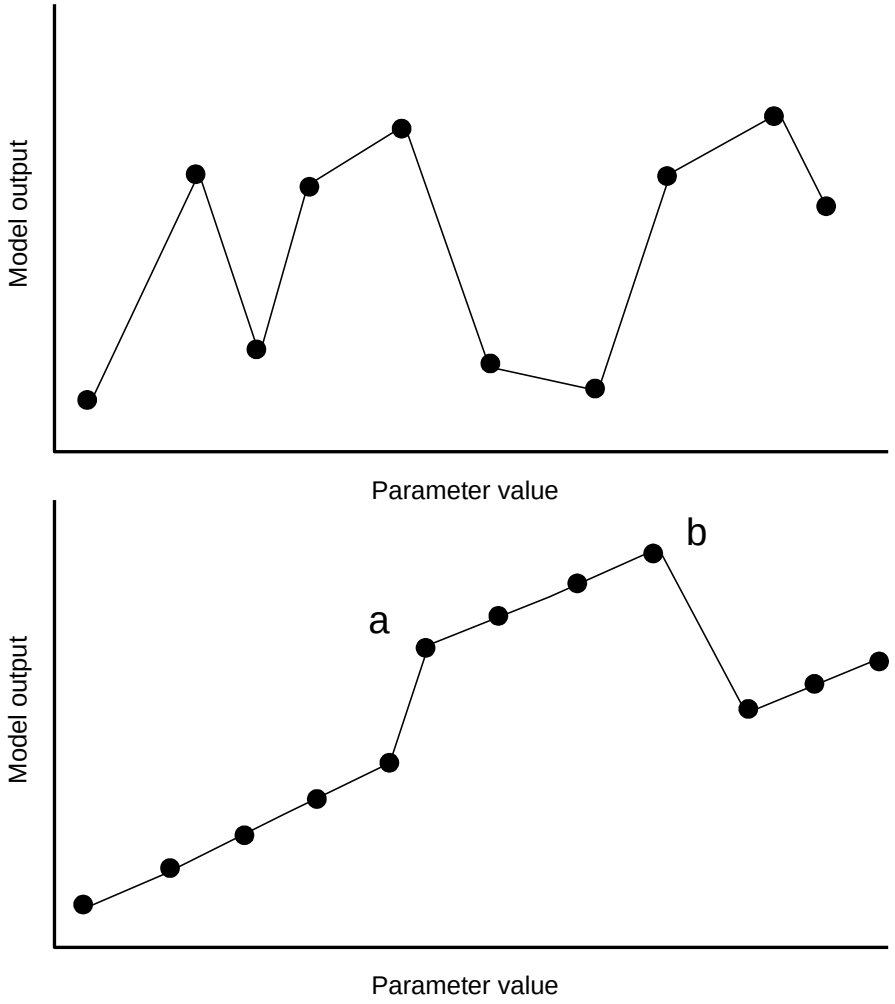


Figure 3.2 Plots of model output vs. parameter value using data read from a JACTEST output file.

Situations described by the second of the above figures may be detectable by PEST where finite-difference derivatives are being computed using a three point (i.e. central) scheme. (As presently programmed, PEST does not implement the following methodology when computing derivatives using a five-point stencil). Under these circumstances PEST has the opportunity to compare the slopes of two neighbouring segments on the curve of model output vs. parameter value. If slopes of these neighbouring segments are markedly different, this indicates that points (a) or (b) of the above graph may have been encountered. If this is the case, the derivative should be computed using only one of the neighbouring segments defined on the basis of the three parameter values, rather than using both of them. Alternatively, it may be better to abandon both slopes in order to approximate the derivative in a way that does not necessarily achieve progress of the parameter estimation process, but does not harm it either.

3.5.7 Split Slope Analysis

"Split slope analysis" is implemented using parameter-group-specific settings available in the "parameter groups" section of the PEST control file. The SPLITTHRESH, SPLITRELDIFF and SPLITACTION variables which implement split slope analysis are all optional. However if one of them is supplied for a particular parameter group, then all of them must be supplied

for that group.

SPLITTHRESH must be zero or greater. If it is set to zero (or omitted), then split slope analysis is not performed for that particular parameter group. If it is set to a positive value, then split slope analysis for all observations for all parameters belonging to the group is potentially undertaken. However it will only be actually undertaken for a particular observation/parameter pair if the absolute value of the slope of at least one of the neighbouring segments employed in three-point derivatives calculation for that observation/parameter pair is greater than the value supplied for SPLITTHRESH. Thus the analysis (and possible omission of individual slope segments) is not undertaken where derivatives are very small.

SPLITRELDIFF defines another threshold. This variable must be provided with a value that is greater than 0.0. If the absolute difference in slopes of neighbouring segments, divided by the smaller of these two slopes, is greater than this threshold, then segment rejection is activated.

The SPLITACTION variable defines the manner in which segment rejection is undertaken. This is a text variable which must be set to either "smaller", "zero" or "previous". If it is set to "smaller", then the slope segment with higher absolute slope value is rejected, and the derivative is taken as the slope of the segment of lesser absolute slope. If it is set to "zero" then the derivative of the current observation with respect to the current parameter is assigned a value of zero for the current iteration. If it is set to "previous", then the derivative obtained in the previous iteration is retained for the present iteration as well.

The following should be noted.

1. If, for a particular parameter group, SPLITACTION is set to "previous", and three-point derivatives computation is being undertaken on the first iteration of the parameter estimation process, the "previous" derivative of the pertinent observation with respect to the pertinent parameter is taken to be zero.
2. The same thing happens if PEST is restarted mid-run (restarts are discussed later in this manual), for on a restart PEST does not read its previous Jacobian matrix.

The SPLITSWH variable which resides in the "control data" section of the PEST control file provides further implementation options for split slope analysis. If SPLITSWH is provided with a zero or negative value, it is ignored. If not, it can be used to determine the point within a PEST run at which split slope analysis begins. Suppose that this is set to 1.10. Then split slope analysis will begin when the following conditions have been met.

1. Central derivatives are computed for at least one parameter group.
2. If the switch to three-point derivatives has just occurred then at least one iteration has elapsed since this switch has taken place. (This gives central derivatives without split slope analysis a chance to lower the objective function.)
3. The ratio of the best objective function achieved during a certain iteration to that achieved in the previous iteration is 1.10.

SPLITSWH can be set to less than 1.0 if desired. For example, if it is set to 0.95, then the use of split slope analysis is triggered when the new-to-old objective function ratio is 0.95 or above.

Experience suggests that use of split slope analysis appears to be moderately successful in allowing the parameter estimation process to progress, even where derivatives are of poor

quality. Settings for SPLITTHRESH, SPLITRELDIFF and SPLITACTION of 1.0E-4, 0.5 and "smaller" appear to work well on most occasions. However use of JACTEST may suggest better settings for a particular occasion.

3.5.8 Model-Calculated Derivatives

As is apparent from the above discussion, calculation of derivatives by finite differences can be both time-consuming and numerically intensive. If a model can calculate derivatives of its outputs with respect to its adjustable parameters itself, then use of these derivatives can benefit the parameter estimation process greatly. This is because of the high level of accuracy with which a model can normally calculate its own derivatives (especially if these are calculated using analytical equations), and the likelihood that a model can undertake such calculations comparatively quickly through modification of the calculations that it undertakes in the normal simulation process. Hence, if they are available, model-calculated derivatives should be used by PEST in preference to finite-difference derivatives.

Chapter 12 of this manual describes the mechanisms through which PEST can receive derivatives calculated internally by a model. In summary, this kind of model-PEST interaction requires that the model generate a file in which these derivatives are recorded. Because the calculation of derivatives by the model may place an extra computational burden on the model's shoulders, it is sometimes necessary that the model be run in a slightly different manner to calculate derivatives from that in which it is run when undertaking normal simulation. This can be accommodated through the use of multiple model command lines and through "PEST-to-model messaging". See Chapter 12 for a discussion of both of these.

As is also described in chapter 12, there will be some situations in which it is possible for a model to calculate some of its derivatives but not others. In cases such as this, PEST can accept those derivatives from the model which the model is capable of calculating, while computing the remaining derivatives itself by the traditional method of finite differences.

3.5.9 Using a Surrogate Model for Derivatives Calculation

Repeated calculation of the Jacobian matrix is normally the most time-consuming part of the inversion process undertaken by PEST. This burden can be eased in a number of ways. One is through parallelisation of model runs, as is undertaken by Parallel PEST and BEOPEST. Another is through defining so-called "super parameters", using singular value decomposition to establish combinations of parameters that are uniquely estimable on the basis of the calibration dataset and restricting the inversion process to estimation of only these. This is the "SVD-assist" methodology supported by PEST; see section 6.4.4 of Doherty (2015) for theoretical details. Another option is to use a simplified model for some or all of the model runs required for filling of the Jacobian matrix. See Burrows and Doherty (2015) for a discussion and examples. This is supported using PEST's "observation re-referencing" functionality described in chapter 14 of this manual.

3.6 The Jacobian Matrix File

Files recorded by PEST are discussed later in this manual. They are also listed in appendix B. A file of particular importance is the Jacobian matrix file. This is named *case.jco* where *case* is the current PEST case name, that is the filename base of the PEST control file (of which the extension is always "*.pst*"). The binary file which holds a Jacobian matrix will often be referred to as a "JCO file" in this manual.

As was mentioned above, PEST calculates a Jacobian matrix during every iteration of the inversion process. Where parameters are improved from the previous iteration, the updated Jacobian matrix is recorded in a JCO file. To save space it is stored in binary, compressed format. Hence it cannot be inspected using a text editor. However it can be converted to ASCII format using the JACWRIT and JCO2MAT utilities provided with PEST. An individual row can be extracted from it, and its contents recorded in ASCII format, using the JROW2VEC utility. The JCO2JCO utility builds a new JCO file corresponding to a new PEST control file whose composition is a subset of that on which basis an original JCO file was recorded. Other utilities support the construction of a large JCO file from component JCO files built on the basis of smaller PEST datasets. See part II of this manual for details.

The JCO file finds many uses. Many of these uses are as important as the model calibration process itself. Hence it is not unusual for a PEST run to be undertaken purely for the purpose of filling a Jacobian matrix and writing a JCO file. This can be achieved if the NOPTMAX variable in the "control data" section of the PEST control file is set to -1 or -2.

Uses to which a JCO file may be put include the following.

- Examination of sensitivities of model outputs with respect to parameters.
- Giving PEST a "head start" in calibrating a model by providing it with a pre-calculated Jacobian matrix. This can be implemented by starting PEST, Parallel PEST or BEOPEST using the "/i" switch.
- Definition of super parameters for an SVD-assist run. The super parameter PEST control file is then built using the SVDAPREP utility.
- As a basis for the many types of linear analysis implemented using the utility programs documented in part II of this manual; these include calculation of
    - parameter identifiability;
    - parameter and predictive uncertainty;
    - parameter contributions to predictive uncertainty;
    - data worth;
    - the effects of model defects.

3.7 The Objective Function

3.7.1 Weights

When run in "estimation" mode, PEST minimises a least-squares objective function. As described by Doherty (2015) this is the sum of squared weighted differences between measurements (or prior information equations) and corresponding model outputs. The difference between a measurement and a model output to which it corresponds is referred to as a residual. Let the *i*'th residual be designated as r_i. Let the weight associated with the *i*'th observation (which may be a prior information equation) be designated as w_i. Then the objective function Φ is calculated as

$$\Phi = \sum (w_i r_i)^2 \qquad (3.7.1)$$

Obviously, if an observation is ascribed a weight of zero, then the residual associated with that observation makes no contribution to the total objective function. You can thus easily take an observation "out of action" by assigning it a weight of zero. This provides a far easier mechanism for removal of an observation from the calibration dataset than that of re-building the PEST control file with this observation absent.

When PEST is run in "predictive analysis" mode, it maximises or minimises a user-specified prediction subject to the constraint that the objective function rises no higher than a specified level.

When PEST is run in "regularisation" mode it calculates two objective functions, namely a measurement objective function and a regularisation objective function. Each of these is defined using weights and residuals in the normal manner. However observations and/or prior information equations are separated into two separate categories, namely measurements on the one hand and regularisation constraints on the other hand; a separate objective function is calculated for each.

When PEST is run in "pareto" mode, one objective function is traded off against another as weights associated with one of them are collectively increased while those associated with the other remain static.

Regardless of PEST's mode of operation, observations and prior information equations can be divided into so-called "observation groups". (Note that the term "observation group" includes prior information groups. Prior information is, after all, "observations" which pertain directly to parameters, or to linear combinations of parameters.) Where observations and prior information equations are grouped in this manner, PEST calculates an objective function for each such group. These objective function components are displayed on the screen and listed in PEST's run record file. This allows you to ensure that no observation group dominates the objective function, or is invisible in the objective function. Either situation can be rectified through adjustment of weights assigned to members of pertinent groups. Utility programs documented in part II of this manual can facilitate this operation. Programs within the PEST Groundwater and Surface Water Utility suites can also be used for inter-group weights adjustment.

3.7.2 Covariance Matrices

As is discussed by Doherty (2015), the use of observation weights in calculating the objective function is based on the premise that the noise associated with observations is independent, i.e. that the "uncertainty" pertaining to any one observation bears no relationship to the "uncertainty" pertaining to any other observation. However if residuals are likely to show consistency over space and/or time for certain observation types, then it may not be appropriate to ascribe statistical independence to observations of these types during the inversion process. In such cases it may be preferable to describe the uncertainties associated with these observations using an observation covariance matrix (or a matrix that is proportional to this matrix), rather than using a set of individual observation weights.

Observation covariance matrices can be particularly useful where prior information is employed in the inversion process, especially if this prior information comprises the "regularisation observations" used by PEST when running in "regularisation" mode. In many cases involving spatially-distributed parameters, individual parameter values, or the differences between individual parameter values, may exhibit some degree of distance-dependent correlation (perhaps expressed by a variogram). This correlation can be included in the inversion process by assigning a covariance matrix that reflects spatial parameter interdependence to pertinent prior information equations. Use of a covariance matrix in this context is far superior than employment of a set of weights that encapsulate a false premise of parameter stochastic independence. Utility programs supplied with the PEST Groundwater Utility suite automate the construction of covariance matrices based on variograms (including "spatially varying variograms").

Observation correlation may be important in other situations as well. Consider, for example, the case where, for a particular groundwater model, the extent of outflow from the groundwater domain into two neighbouring reaches of a stream comprises part of the calibration dataset. Consider also (as often occurs in practice) that the total outflow into both of the neighbouring reaches can be more accurately measured (and possibly computed by the model) than the outflow into each individual reach. Nevertheless it may be considered desirable for a particular model application that the model be calibrated using both of the reach outflows individually rather than the total outflow. Because the uncertainties associated with the individual reach outflow measurements will not be independent (for a positive error in one is likely to be complemented by a negative error in the other), it would be better to assign a covariance matrix to these observations which reflects their interdependence, than to ignore this interdependence through the assignment of separate, individual weights to these observations when undertaking the inversion process.

Let C(**ε**) denote a covariance matrix associated with a group of observations. (As discussed extensively by Doherty (2015) this matrix is more appropriately designated as C(**k**) where it is applied to a group of prior information equations which encapsulate expert knowledge of parameter variability.) Then the component of the objective function that is contributed by this group of observations or prior information equations is calculated as

$$\Phi = \mathbf{r}^t \mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{r} \qquad (3.7.2)$$

Note that a covariance matrix must be positive definite, this guaranteeing that its contribution to the total objective function as calculated using (3.7.2) is always positive.

The design of PEST is such that if PEST is supplied with a covariance matrix, that matrix must pertain to a specific observation group. Because prior information items can also be assigned to one or more observation groups, this allows a covariance matrix to be supplied for a group of prior information items, just as it can be supplied for a group of observations.

More than one covariance matrix can be supplied to PEST for use in the inversion process. In fact a covariance matrix can be supplied for every observation group. However, more often than not, it will be supplied for only one or two such groups, with weights being used for the remainder of the groups.

4. The PEST Control File

4.1 Introduction

4.1.1 General

This section discussed the PEST control file, and provides specifications for many of the variables which it features. However not every variable appearing in the PEST control file is discussed in the present chapter. Some variables, and some entire sections of the PEST control file, are discussed later in this manual where the aspects of PEST's functionality to which they pertain are presented in detail.

4.1.2 The Role of the PEST Control File

Once template and instruction files have been prepared for a particular case, a PEST control file must be prepared which "brings it all together". Unlike template and instruction files, for which there is no naming convention, there are some conventions associated with the name of the PEST control file. In particular, the file must have an extension of "*.pst*". Its filename base is referred to as the PEST "case name" herein; PEST uses this same filename base for the files which it generates in the course of its run. Let *case* represent this case name. The PEST control file is therefore named *case.pst*. As it runs, PEST generates a set of files which all have this same filename base. These include *case.rec* (the run record file), *case.par* (best parameter values), *case.rst* (contains restart information), *case.jco* (the Jacobian matrix file) and other files listed in appendix B which are discussed in the following chapter. Some special purpose PEST input files discussed elsewhere in this manual must have this same filename base, for example *case.rmf* (Parallel PEST run management file) and *case.hld* (a "parameter hold file" which supports PEST's user-intervention capabilities).

The PEST control file can be built in a number of ways. It can easily be prepared using a text editor following the directions provided in this chapter. Alternatively, you can use the PESTGEN utility (see part II of this manual) to generate a PEST control file for your current case which uses default input variables; this file can then be modified as you see fit using a text editor. Another alternative is to employ those programs from the PEST Groundwater and Surface Water Utilities which facilitate construction of a PEST control file. These have been designed to construct complex PEST input datasets which may feature hundreds, or even thousands, of parameters and observations. A further alternative is to leave all of the hard work to the graphical user interface of a model which supports PEST. In that case you may never need to see a PEST control file. However you will need to know the roles played by various PEST control variables, as you will probably have to choose their values through dialogue boxes featured in those interfaces.

In all of these cases the PEST control file can be checked for correctness and consistency using the PESTCHEK utility discussed in part II of this manual.

4.1.3 Some Specifications

The first line of a PEST control file must be comprised only of the letters "pcf" denoting "PEST control file. This text is case-insensitive (as is all other text featured in a PEST control file).

The PEST control file consists of integer, real and character variables separated by spaces.

The value of each variable must be separated from its neighbour by at least one space. Real numbers can be supplied with the minimum precision necessary to represent their values; the decimal point does not need to be included if it is redundant. Note, however, that all real numbers are stored internally by PEST as double precision numbers. If exponentiation is required, this can be accomplished using the "e" symbol.

Complete specifications of the PEST control file are set out in figure A1.1 of Appendix A where variables are referenced by name. Careful note should be taken of where each variable resides on its particular line of the PEST control file. Not all variables need to be present on all occasions; variables which can be omitted are enclosed in square brackets in figure A1.1. The tables following figure A1.1 provide a short description of the role of each variable. More detailed descriptions are provided in this chapter, and in chapters that follow. For some parameter and derivative-related variables, detailed descriptions have already been provided in the previous chapter.

As was mentioned in the preceding chapter, the PEST control file is subdivided into sections, each of which has a section header whose name begins with the "*" character; a space must separate this character from the text which follows it. This text must be exactly as set out in figure A1.1. Some sections of the PEST control file are optional, these being required only if the functionality which they govern is to be implemented on a particular PEST run. Three sections are specific to a certain mode of PEST operation, these being the "predictive analysis", "regularisation" and "pareto" sections.

The present chapter focusses on "estimation" mode. Other modes of PEST operation are fully discussed in later chapters. Similarly, some advanced and specialized aspects of PEST's operation will also be left until later chapters. Variables and sections of the PEST control file which pertain to those aspects of PEST functionality will be described in detail in those chapters.

An example of a PEST control file is presented in figure 4.1. Note that this is a particularly simple PEST control file from which many optional variables have been omitted.

```
pcf
* control data
restart estimation
5 19 2 2 3
2 3 single point
10.0 -3.0 0.3 0.03 10
3.0 3.0 0.001
0.1
50 0.005 4 4 0.005 4
1 1 1
* parameter groups
ro  relative 0.01 0.0001 switch 2.0 parabolic
h   relative 0.01 0.0001 switch 2.0 parabolic
* parameter data
ro1  fixed factor  0.5   .1  10     ro   1.0 0.0
ro2  log   factor  5.0   .1  10     ro   1.0 0.0
ro3  tied  factor  0.5   .1  10     ro   1.0 0.0
h1   none  factor  2.0   .05 100    h    1.0 0.0
h2   log   factor  5.0   .05 100    h    1.0 0.0
ro3  ro2
* observation groups
obsgp1 cov.mat
obsgp2
prgp1
* observation data
ar1  1.21038  1.0  obsgp1
ar2  1.51208  1.0  obsgp1
```

```
ar3  2.07204  1.0  obsgp1
ar4  2.94056  1.0  obsgp1
ar5  4.15787  1.0  obsgp1
ar6  5.7762   1.0  obsgp1
ar7  7.7894   1.0  obsgp1
ar8  9.99743  1.0  obsgp1
ar9  11.8307  1.0  obsgp2
ar10 12.3194  1.0  obsgp2
ar11 10.6003  1.0  obsgp2
ar12 7.00419  1.0  obsgp2
ar13 3.44391  1.0  obsgp2
ar14 1.58279  1.0  obsgp2
ar15 1.1038   1.0  obsgp2
ar16 1.03086  1.0  obsgp2
ar17 1.01318  1.0  obsgp2
ar18 1.00593  1.0  obsgp2
ar19 1.00272  1.0  obsgp2
* model command line
model.bat
* model input/output
ves1.tpl a_model.in1
ves2.tpl a_model.in2
ves1.ins a_model.ot1
ves2.ins a_model.ot2
ves3.ins a_model.ot3
* prior information
pi1  1.0 * h1 = 1.0 3.0 prgp1
pi2  1.0 * log(ro2) + 1.0 * log(h2) = 2.6026 2.0 prgp1
```

Figure 4.1 An example of a PEST control file.

4.1.4 Comments, Blank Lines and "++" Lines

PEST, BEOPEST and many of the PEST-support utility programs which are documented in part II of this manual tolerate the presence of the following items in a PEST control file:

- blank lines;
- comments;
- lines that begin with the character string "++".

Lines that begin with "++" are used for the insertion of variables which control the operation of the PEST++ suite of programs.

Comments can be placed on their own line. Alternatively, they can be placed at the end of a line which provides PEST control data. In either case, a comment follows a "#" character. Note, however, that this character is not construed as denoting the presence of an ensuing comment under any of the following circumstances:

- it is not prededed by a space, tab or the beginning of a line;
- it is part of a string that is enclosed in quotes.

These exceptions preclude mis-construing the presence of the "#" character in a filename as the start of a comment.

Some of the older utilities that are documented in part II of this manual will not tolerate the presence of blank lines, "++" lines or comments. All of these items can be removed from a PEST control file using the PSTCLEAN utility.

4.2 Control Data Section

4.2.1 General

Variables appearing in the "control data" section of the PEST control file are shown in figure 4.2 (which is reproduced from figure A1.1 of appendix A). These will now be discussed line by line. However discussion of some optional variables will be left until later sections of this manual where a broader discussion of the capabilities to which they pertain will be presented.

```
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM] [DERZEROLIM]
NTPLFLE NINSFLE PRECIS DPOINT [NUMCOM JACFILE MESSFILE] [OBSREREF]
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE] [LAMFORGIVE] [DERFORGIVE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND] [ABSPARMAX]
PHIREDSWH [NOPTSWITCH] [SPLITSWH] [DOAUI] [DOSENREUSE] [BOUNDSCALE]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN] [PHIABANDON]
ICOV ICOR IEIG [IRES] [JCOSAVE] [VERBOSEREC] [JCOSAVEITN] [REISAVEITN] [PARSAVEITN] [PARSAVERUN]
```

Figure 4.2 Variables in the "control data" section of a PEST control file.

4.2.2 First Line

The first line of the "control data" section of the PEST control file must contain the string "* control data". Note that the "*" character is followed by a space.

4.2.3 Second Line

RSTFLE

This character variable must be assigned one of two values, namely "restart" or "norestart". If it is supplied as "restart", PEST "leaves tracks" as it runs, dumping the contents of its arrays to binary files, and continuously updating those files, so that its execution can be re-commenced later if it is intentionally or accidently halted. PEST execution can be restarted using a variety of switches, each of which instructs PEST to behave differently as it re-commences the inversion process. See later for details. See appendix B for a listing of the files which hold PEST's restart data; note that these files can become large where an inverse problem employs many parameters and many observations.

If the RSTFLE variable is set to "norestart", PEST will not leave tracks; hence a later re-commencement of execution is impossible.

PESTMODE

This character variable must be supplied as "estimation", "prediction", "regularisation" or "pareto". For all but "estimation", a section must be added to the bottom of the PEST control file containing variables which govern its operation when working in that mode. If such a section is present and PEST is not running in that mode, then that section is ignored.

4.2.4 Third Line

NPAR

This is the total number of parameters featured in the current PEST case, including adjustable, fixed and tied parameters; NPAR must be supplied as an integer.

NOBS

This integer variable represents the total number of observations featured in the current case. Note that, when counting the number of observations, dummy observations (see chapter 2) that may be featured in one or a number of instruction files are ignored.

NPARGP

This is the number of parameter groups. Recall from the previous chapter that the variables which govern the operation of finite-difference derivatives are assigned to parameter groups. Parameter grouping can also be of importance where the ADDREG1 utility described in part II of this manual is employed to add preferred value Tikhonov regularisation to a PEST control file.

NPARGP is an integer variable.

NPRIOR

NPRIOR, another integer variable, is the number of articles of prior information that are included in the parameter estimation process. If there are no articles of prior information, NPRIOR must be zero.

If PEST is running in "estimation" mode, then you should ensure that the number of adjustable parameters is less than or equal to the number of observations for which there are non-zero weights plus the number of articles of prior information for which there are non-zero weights. If this is not the case, then the inverse problem cannot possibly have a unique solution. Furthermore, unless solution of this nonunique inverse problem is sought using singular value decomposition or LSQR, PEST may make little progress in lowering the objective function because the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix appearing in equation 5.2.9 of Doherty (2015) will be singular and hence non-invertible.

Sadly, nonuniqueness is the rule rather than the exception when calibrating environmental models. PEST is not troubled by parameter nonuniqueness if Tikhonov regularisation is introduced to the inverse problem, and if singular value decomposition or LSQR is used in solution of that problem. Because of the prevalence of nonuniqueness in environmental model calibration, this should be done as a matter of course. (The outnumbering of observations by parameters does not create a numerical problem if singular value decomposition or LSQR are used to solve the inverse problem, despite the noninvertability of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$.)

NOBSGP

NOBSGP, another integer variable, is the number of observation groups featured in the PEST control file. Each observation and each prior information equation must be assigned to an observation group (they can all be assigned to the same group if desired). When PEST evaluates the total objective function it also evaluates the contribution made to this total by each observation group.

MAXCOMPDIM

MAXCOMPDIM is an optional integer variable. It is used to activate compressed internal storage of the Jacobian matrix by PEST. This can slow PEST execution. At the same time it can allow PEST to undertake very highly parameterized inversion wherein adjustable parameters may number in the tens of thousands. This is further discussed in section 15.4,

where the role of the MAXCOMPDIM variable is fully explained.

DERZEROLIM

Like MAXCOMPDIM, DERZEROLIM is optional. If supplied, it supports the use of MAXCOMPDIM, defining a threshold below which the value of an element of the Jacobian matrix can be considered to be zero, and hence omitted from compressed storage.

4.2.5 Fourth Line

NTPLFLE

This is an integer variable, informing PEST of the number of model input files which contain parameters; PEST must write each of these files prior to a model run. As there must be one template file for each such model input file, NTPLFLE is also equal to the number of template files which PEST must use in writing the current parameter set.

A model may have many input files; however PEST is concerned only with those which it needs to rewrite prior to each model run, i.e. those for which there are template files. As explained later, a single template file may, under some circumstances, be used to write more than one model input file. In such a case you must count each template file - model input file pair separately in determining NTPLFLE.

NINSFLE

This is the number of instruction files. There must be one instruction file for each model output file containing model-generated observations which PEST reads. (In some circumstances, a single model output file may be read by more than one instruction file; however each instruction file - model output file pair is counted separately in determining NINSFLE).

PRECIS

PRECIS is a character variable which must be either “single” or “double”. If it is supplied to PEST as “single”, PEST writes parameters to model input files using single precision protocol; hence parameter values will never be greater than 13 characters in length (even if the parameter space allows for a greater length) and the exponentiation character is “e”. If PRECIS is supplied as “double”, parameter values are written to model input files using double precision protocol; the maximum parameter value length is 23 characters and the exponentiation symbol is “d”. See section 2.2.6.

DPOINT

This character variable must be either “point” or “nopoint”. If DPOINT is provided with the value “nopoint” PEST will omit the decimal point from representations of parameter values on model input files if the decimal point is redundant, thus making room for the use of one extra significant figure. If DPOINT is supplied as “point” (which is normally recommended), PEST will ensure that the decimal point is always present. See section 2.2.6.

NUMCOM, JACFILE and MESSFILE

These variables are used to control the manner in which PEST can obtain derivatives directly from the model if these are available; see chapter 12. For normal operation these should be set to 1, 0 and 0 respectively. Alternatively, all of them can be omitted. However if one of

them is cited in the PEST control file then all of them must be cited. Also if a value is supplied for NUMCOM, then a DERCOM value must be supplied for all parameters in the "parameter data" section of the PEST control file; If NUMCOM is 1, then all of these DERCOMs must also be 1.

OBSREFEF

Observation re-referencing is activated by adding the string "obsreref" to the fourth line of the "control data" section of the PEST control file. The "obsreref" string can be placed anywhere on this line. Observation re-referencing can be de-activated (the default condition) by omitting this string, or by placing the string "noobsreref" on this line. Observation re-referencing is discussed in chapter 14 of this manual.

4.2.6 Fifth Line

RLAMBDA1

This real variable is the initial Marquardt lambda. As discussed in the previous section, each iteration of the inversion process undertaken by PEST is divided into two parts. The first part comprises calculation of the Jacobian matrix. The second part is devoted to the testing of parameter upgrades that are calculated using different values of the Marquardt lambda. The role of the Marquardt lambda is described in section 5.4.2 of Doherty (2015).

During any one iteration of the inversion process PEST attempts parameter improvement using a number of different Marquardt lambdas. The starting value of the Marquardt lambda during any one iteration is inherited from the previous iteration; it is generally somewhat lower than that which allowed calculation of the lowest objective function during that iteration. Over the course of the entire inversion process the Marquardt lambda should generally fall. However it may rise if an inverse problem is poorly posed.

RLAMBDA1 is the initial value of the Marquardt lambda. This informs PEST of the first value that it should test during the first iteration of the inversion process. A value of 10.0 is appropriate in most cases. However if PEST complains about the normal matrix being singular you may need to raise it. (A better alternative would be to add regularisation to the PEST control file and/or employ singular value decomposition or LSQR as a solution mechanism for the inverse problem.)

RLAMFAC

RLAMFAC, a real variable, is the factor by which PEST adjusts the Marquardt lambda as it tests different values of this variable for their efficacy in lowering the objective function. RLAMFAC must be greater than 1.0; a value of 2.0 seems to work well on many occasions. When PEST reduces lambda it divides by RLAMFAC; when it increases lambda it multiplies by RLAMFAC. PEST reduces lambda if it can. However if the inverse problem is nonunique and is unsupplemented by regularisation or by use of singular value decomposition or LSQR as a solution device, or if a reduction in lambda does not lower the objective function, PEST has no choice but to increase lambda.

Alternatively, the Marquardt lambda can be given a negative value less than -1.0. Suppose that it is given a value of -3.0. Then PEST will adjust the lambda adjustment factor during each iteration of the inversion process so that lambda can achieve a value of 1.0 with three adjustments (if PEST decides to move it in this direction). This allows rapid adjustment of the Marquardt lambda if this is suddenly required, as can happen if local parameter insensitivity

promulgates sudden problem ill-posedness.

More specifically, suppose that RLAMFAC is supplied as $-r$ where r is positive. Let λ be the value of the Marquardt lambda at the beginning of a particular iteration, this being inherited from the previous iteration as described above. Then a Marquardt lambda adjustment factor f for use in the current iteration is calculated as follows.

$$f = \min[\lambda^{1/r}, 2.0] \quad \text{if } \lambda > 1.0 \qquad (4.2.1a)$$

$$f = \min[(1/\lambda)^{1/r}, 2.0] \quad \text{if } \lambda < 1.0 \qquad (4.2.1b)$$

$$f = 2.0 \quad \text{if } \lambda = 1.0 \qquad (4.2.1c)$$

If supplying a negative value for RLAMFAC it is important to note that the higher the absolute value assigned to a negative RLAMFAC, the smaller will be the actual iteration-specific Marquardt lambda adjustment factor. This is opposite to the case for a positive setting of RLAMFAC. In both cases however, the absolute value of RLAMFAC must be greater than 1.0.

An RLAMFAC value of -3.0 works well on most occasions.

PHIRATSUF

During any one iteration of the inversion process, PEST may calculate a parameter upgrade vector using a number of different Marquardt lambdas. First it lowers lambda and, if this is unsuccessful in lowering the objective function, it then raises lambda. If, at any stage, it calculates an objective function which is a fraction PHIRATSUF or less of the starting objective function for that iteration, PEST considers that the goal of the current iteration has been achieved and moves on to the next iteration. Thus PEST will commence iteration i+1 if, at any stage during iteration i

$$\Phi_l^j/\Phi_{i-1} \leq \text{PHIRATSUF} \qquad (4.2.2)$$

where Φ_{i-1} is the lowest objective function calculated for iteration i-1 (and hence the starting value for iteration i) and Φ_l^j is the objective function corresponding to a parameter set calculated using the j'th Marquardt lambda tested during iteration i.

PHIRATSUF (which stands for "phi ratio sufficient") is a real variable for which a value of 0.3 is mostly appropriate. If it is set too low, model runs may be wasted in search of an objective function reduction which it is not possible to achieve, given the linearity approximation upon which the inversion equations described in Doherty (2015) are based. If it is set too high, PEST may not be given the opportunity of refining lambda in order that its value continues to be optimal as the inversion process progresses.

PHIREDLAM

If a new/old objective function ratio of PHIRATSUF or less is not achieved as the effectiveness of different Marquardt lambdas in lowering the objective function is tested, PEST must use some other criterion in deciding when it should move on to the next iteration. This criterion is partly provided by the real variable PHIREDLAM.

The first lambda that PEST employs in calculating the parameter upgrade vector during any particular iteration is the lambda inherited from the previous iteration, possibly reduced by a factor of RLAMFAC (unless it is the first iteration, in which case RLAMBDA1 is used). Unless, through the use of this lambda, the objective function is reduced to less than PHIRATSUF of its value at the beginning of the iteration, PEST then tries another lambda,

less by a factor of RLAMFAC than the first. If the objective function is lower than for the first lambda (and still above PHIRATSUF of the starting objective function), PEST reduces lambda yet again; otherwise it increases lambda to a value greater by a factor of RLAMFAC than the first lambda for the iteration. If, in its attempts to find a more effective lambda by lowering and/or raising lambda in this fashion, the objective function begins to rise, PEST accepts the lambda and the corresponding parameter set giving rise to the lowest objective function for that iteration, and moves on to the next iteration. Alternatively if the relative reduction in the objective function between the use of two consecutive lambdas is less than PHIREDLAM, PEST takes this as an indication that it is probably more efficient to begin the next iteration than to continue testing the effect of new Marquardt lambdas. Thus if

$$(\Phi_i^{j-1} - \Phi_i^{j})/\Phi_i^{j-1} \leq \text{PHIREDLAM} \qquad (4.2.3)$$

where Φ_i^j is the objective function value calculated during optimisation iteration i using the j'th trial lambda, PEST moves on to iteration i+1.

A suitable value for PHIREDLAM is between 0.01 and 0.05. If it is set too large, the criterion for moving on to the next iteration is too easily met and PEST is not given the opportunity of adjusting lambda to its optimal value for that particular stage of the parameter estimation process. On the other hand if PHIREDLAM is set too low, PEST will test too many Marquardt lambdas on each iteration of the inversion process when it would be better off starting a new iteration. (Note that where model runs are parallelised, the testing of too many Marquardt lambdas may not be so much of an issue as it keeps otherwise idle processors busy.)

NUMLAM

This integer variable places an upper limit on the number of lambdas that PEST will test during any one iteration. It should normally be set between 5 and 10 (normally closer to 10); however if RLAMBDA1 is set to zero (which is not recommended) it must be set to 1.

JACUPDATE

The Broyden Jacobian update procedure is described in section 5.4.2 of Doherty (2015). It provides a mechanism for improving the Jacobian matrix based on model outputs calculated during model runs undertaken for the purpose of testing parameter upgrades calculated using different values of the Marquardt lambda.

If JACUPDATE is omitted from the fifth line of the "control data" section of the PEST control file, or if it is set to zero, Broyden updating of the Jacobian matrix does not occur. However if it is set to *n*, where *n* is a positive integer, Broyden updating will be undertaken during the second part of every PEST iteration following the first *n* attempts to upgrade parameters using *n* different values of the Marquardt lambdas. Alternatively, it can be set to 999. In this case the Jacobian matrix is updated after the first parameter upgrade attempt, and every parameter upgrade attempt thereafter.

Experience indicates that Broyden Jacobian updating can work well on most occasions, but on some occasions does not. At the time of writing it is difficult to predict when it is likely to work well and when the opposite is likely to occur. The "safest" option is not to activate it as it can, under some circumstances, actually hinder the progress of the inversion process. Still, these occasions appear to be relatively rare, and in the majority of cases it is worth a try.

LAMFORGIVE

Sometimes, when a model is provided with a particular set of parameters, it fails to run to completion. Its output file is not then written. PEST reports this error condition and ceases execution with an appropriate error message. This situation is most likely to arise during those phases of the parameter estimation process where PEST is altering the value of the Marquardt lambda and testing new upgraded parameter values; this is the stage of the inversion process where parameters provided to the model are most different from their predecessors, and hence where model run failure is most likely to occur.

In most deployment contexts, a better course of action for PEST to take is to ignore the failed model run and attempt calculation of further parameter upgrades using other values of the Marquardt lambda. This response to model run failure can be instigated using the LAMFORGIVE variable.

LAMFORGIVE should be placed on the fifth line of the "control data" section of the PEST control file following the JACUPDATE variable if it is present, or the NUMLAM variable if JACUPDATE is absent. (Actually LAMFORGIVE can be placed in front of the JACUPDATE variable if desired.) It must be supplied as "lamforgive" (to activate model run failure forgiveness) or "nolamforgive" (to de-activate model run failure forgiveness). If it is omitted, a default value of "nolamforgive" is used by PEST.

Suppose that LAMFORGIVE is supplied as "lamforgive". If, after a few parameter upgrade attempts have been made (normally four), it is found that no lambda value leads to an upgraded parameter set that allows the model to complete execution, PEST takes appropriate action. It either switches to central derivatives calculation, or ceases execution altogether (with an appropriate error message).

Sometimes a user may wish to know the parameters that caused model run failure. These are stored in parameter value file format (see section 5 of this manual) in a sequence of files named *###error.par.1*, *###error.par.2*, etc. Files of this type resulting from a previous run are deleted at the beginning of each new PEST run. Counting starts afresh when PEST starts afresh.

DERFORGIVE

DERFORGIVE plays a similar role to LAMFORGIVE. However DERFORGIVE governs PEST's behaviour when model runs are being undertaken for the purpose of finite-difference derivatives calculation.

Like LAMFORGIVE, the DERFORGIVE variable (if provided) must be placed on the fifth line of the "control data" section of the PEST control file somewhere after NUMLAM. It must be supplied as either "derforgive" or "noderforgive". If it is omitted, a value of "noderforgive" is assumed. A value of "noderforgive" preserves normal PEST behaviour following model run failure during derivatives calculation. Under these circumstances PEST makes one attempt to repeat the model run using identical parameter values. If that run fails, PEST ceases execution with an appropriate error message. In contrast, when DERFORGIVE is set to "derforgive" and a model run failure occurs, PEST writes a short message to the screen reporting the absence of each missing model output file that it attempts to read. The partial derivatives of model outputs with the respect to the pertinent incrementally-varied parameter are then assigned a value of zero.

Note that if BEOPEST is being used, the slave which encounters the failed model run, rather than the BEOPEST master, reports the missing model output files to the screen. Note also

that Parallel PEST does not implement DERFORGIVE functionality; if Parallel PEST is run with DERFORGIVE set to "derforgive" it will cease execution with an appropriate error message.

4.2.7 Sixth Line

RELPARMAX, FACPARMAX and ABSPARMAX(N)

PEST provides three input variables which can be used to limit parameter adjustments during any iteration of the inversion process; see section 5.4.2 of Doherty (2015) for a discussion of the need for parameter adjustment limiting. These variables are named RELPARMAX, FACPARMAX and ABSPARMAX(*N*). The last of these is actually an array of variables.

As described in section 3.4.8 of this document, RELPARMAX is the maximum relative change that a parameter is allowed to undergo between optimisation iterations, whereas FACPARMAX is the maximum factor change that a parameter is allowed to undergo. ABSPARMAX(*N*) is the maximum absolute change that parameters assigned to change-group *N* are allowed to undergo. Any particular parameter can be subject to only one of these constraints; i.e. a particular parameter must be either relative-limited, factor-limited or absolute(*N*)-limited in its adjustment. Parameters are denoted as either relative-limited, factor-limited or absolute(*N*)-limited through the character variable PARCHGLIM associated with each parameter in the "parameter data" section of the PEST control file.

The relative change in parameter *b* between iterations *i*-1 and *i* is defined as

$$(b_{i-1} - b_i)/b_{i-1} \qquad (4.2.4)$$

If parameter *b* is relative-limited, the absolute value of this relative change must be less than RELPARMAX. If a parameter upgrade vector is calculated such that the relative adjustment for one or more relative-limited parameters is greater than RELPARMAX, the magnitude of the upgrade vector is reduced such that this no longer occurs.

The factor change for parameter *b* between iterations *i*-1 and *i* is defined as

$$\begin{aligned} &b_{i-1}/b_i && \text{if } |b_{i-1}| > |b_i|, \text{ or} \\ &b_i/b_{i-1} && \text{if } |b_i| > |b_{i-1}| \end{aligned} \qquad (4.2.5)$$

If parameter *b* is factor-limited, this factor change (which either equals or exceeds unity according to equation 4.2.5) must be less than FACPARMAX. If a parameter upgrade vector is calculated such that the factor adjustment for one or more factor-limited parameters is greater than FACPARMAX, the magnitude of the upgrade vector is reduced such that this no longer occurs.

The absolute change for parameter *b* between iterations *i*-1 and *i* is defined as

$$|b_i - b_{i-1}| \qquad (4.2.6)$$

If parameter *b* is absolute-limited because PARCHGLIM for this variable has been designated as "absolute(*N*)" where $0 \leq N \leq 10$, this absolute change must be less than ABSPARMAX(*N*). If a parameter upgrade vector is calculated such that the absolute change for one or more absolute(*N*)-limited parameters is greater than ABSPARMAX(*N*), the magnitude of the upgrade vector is reduced such that this no longer occurs.

Whether a parameter should be relative-limited, factor-limited or absolute-limited depends on the parameter. Emplacement of a relative of factor limit works well on most occasion;

requirements for absolute parameter change limits are relatively rare.

Use of a relative limit requires caution however, as a parameter can be reduced from its current value right down to zero for a relative change of only 1; as described in section 3.4.8 it may then take many iterations to re-adjust the value of the parameter upwards, this introducing serious inefficiencies to the parameter estimation process. If you wish to limit the extent of its downward movement during any one iteration to less than this, you may wish to set RELPARMAX to, for example, 0.5; however this may unduly restrict its upward movement. It may be better to declare the parameter as factor-limited. If so, a FACPARMAX value of, say 5.0, would limit its downward movement on any one iteration to 0.2 of its value at the start of the iteration and its upward movement to 5 times its starting value. This may be a more sensible approach for many parameters. Alternatively, provide the parameter with a non-zero OFFSET value and adjust its upper and lower bounds such that it never becomes zero, or nearly zero.

It is important to note that a factor limit will not allow a parameter to change sign. Hence if a parameter must be free to change sign in the course of the inversion process, it must be relative-limited or absolute(*N*)-limited; furthermore, if it is relative-limited, RELPARMAX must be set to greater than unity or the change of sign will be impossible. Thus the PESTCHEK utility (see part II of this manual) will not allow you to declare a parameter as factor-limited, or as relative-limited with a relative limit of less than 1, if its upper and lower bounds are of opposite sign. Similarly, if a parameter's upper or lower bound is zero, it cannot be factor-limited and RELPARMAX must be at least unity.

Suitable values for RELPARMAX and FACPARMAX vary between cases. For highly non-linear problems, these values are best set low. If they are set too low, however, the inversion process can be very slow. An inspection of the PEST run record will often reveal whether you have set these values too low, for PEST records the maximum parameter factor and relative changes on this file at the end of every iteration. If these changes are always at their upper limits and the estimation process is showing no signs of instability, it is quite possible that RELPARMAX and/or FACPARMAX could be increased (or that better regularisation be provided for insensitive parameters).

If you are unsure of how to set these parameters, a value of 10.0 for each of them is often suitable. In cases of extreme nonlinearity, be prepared to set them lower, however. But note that FACPARMAX can never be less than 1; RELPARMAX can be less than 1 as long as no parameter's upper and lower bounds are of opposite sign.

The best value for ABSPARMAX(*N*) is context specific; hence no general advice can be providing on assigning a value to this variable. If no parameters are designated as absolute(*N*)-limited in the "parameter data" section of the PEST control file, then ABSPARMAX(*N*) does not need to feature on the sixth line of the "control data" section of the PEST control file. If supplied, it is written as "absparmax(*n*) = *value*". This string can be placed anywhere on this line; however it is recommended that it follow FACORIG. Three absolute limits feature in the following example of line 6 of the "control data" section of a PEST control file.

```
10.0 10.0 0.001 absparmax(3)=1  absparmax(4)=0.5  absparmax(2)=1.0
```

Figure 4.3 An example of line 6 of the "control data" section of a PEST control file.

If any parameter is absolute(*N*)-limited then the maximum absolute(*N*) change is recorded on the PEST control file together with maximum factor and relative changes at the end of each

PEST iteration.

FACORIG

If, in the course of the inversion process, a parameter becomes very small, the relative or factor limit to subsequent adjustment of this parameter may severely hamper its growth back to higher values, resulting in very slow convergence to an objective function minimum. Furthermore, for the case of relative-limited parameters which are permitted to change sign, it is possible that the denominator of equation 4.2.4 could become zero.

To obviate these possibilities, choose a suitable value for the real variable, FACORIG. If the absolute value of a parameter falls below FACORIG times its original value, then FACORIG times its original value is substituted for the denominator of equation 4.2.4. For factor-limited parameters, a similar modification is introduced to equation 4.2.5. Thus the constraints that apply to a growth in absolute value of a parameter are lifted when its absolute value has become less than FACORIG times its original absolute value. However, where PEST wishes to reduce the parameter's absolute value even further, factor-limitations are not lifted; nor are relative limitations lifted if RELPARMAX is less than 1. FACORIG is not used to adjust limits for log-transformed parameters.

FACORIG must be greater than zero. A value of 0.001 is often suitable.

IBOUNDSTICK

Operation of the IBOUNDSTICK variable is described in section 3.4.5 of this manual. It can be used to reduce the number of model runs required per iteration by permanently freezing at their bounds parameters which have encountered their bounds and seem reluctant to move from there. If IBOUNDSTICK is supplied as zero, or omitted altogether, PEST's handling of bounds is unchanged from its normal operation as described in section 3.4.4. However if it is set to *n* (where *n* is a positive integer), PEST will permanently "glue" a parameter to its upper or lower bound if that parameter has resided there for *n* optimisation iterations. Once a parameter is "glued" to one of its bounds it will never move again, for PEST will no longer include this parameter in its upgrade vector calculations. Nor will it calculate derivatives with respect to this parameter, thus reducing the number of model runs required per iteration

You should carefully note the following points regarding the use of IBOUNDSTICK.

1. If IBOUNDSTICK is set to 1, then parameters will be glued to their bounds from the moment that they strike them (beginning at the iteration immediately following the bounds encounter). Thus if the initial value of a parameter is at its upper bound and IBOUNDSTICK is set to 1, then the parameter will be glued to its bound from the very first iteration.
2. If IBOUNDSTICK is set to 2, then 1 complete iteration will elapse (in which the parameter is free to move back into allowed parameter space) since the iteration at which it encountered its bound before that parameter is glued to its bound.

If used, a good setting for IBOUNDSTICK is 2 to 4. However experience has shown that use of the IBOUNDSTICK variable is generally not advisable. Hence it should normally be omitted or set to zero. This is because, for nonlinear models, the sensitivities of parameters can depend not just on their own values, but on those of others. Hence a parameter may have a propensity to reside at its bound during certain stages of the inversion process but not at other stages. Another problem with use of bounds sticking functionality, is that PEST loses the capacity to calculate statistics such as composite sensitivities and the post-calibration

parameter covariance matrix. Furthermore the derivative of the parameter is not included in the Jacobian matrix file. Hence the Jacobian matrix must be re-computed before linear analysis utilities described in part II of this manual can be used. These problems can be overcome by adopting the following procedure on completion of a PEST run in which bounds sticking is activated.

1. Use the PARREP utility (see part II of this manual) to create a new PEST control file based on the original control file, but with optimised parameter values replacing the initial values used for the previous run.
2. Remove IBOUNDSTICK from this new file, or set it to greater than 1.
3. Set the NOPTMAX control variable to -1 or -2 in the new PEST control file.
4. Run PEST to obtain a JCO file; this contains the Jacobian matrix calculated on the basis of previously-estimated parameters.

If the IBOUNDSTICK variable is provided on the sixth line of the PEST control file, its value should follow that of FACORIG. As stated above, omission of this variable is generally the preferred option.

UPVECBEND

The presence of the UPVECBEND variable on the sixth line of the PEST control file is optional. However, if supplied, it must immediately follow IBOUNDSTICK.

As described in section 3.4.5, if UPVECBEND is set to 1, PEST will not shorten the parameter upgrade vector to ensure that parameter factor, relative and absolute change limits are respected. Instead it will "bend" this vector to allow imposition of limits on pertinent parameters while allowing unrestricted upgrade of parameters that, according to the parameter upgrade vector, are not required to undergo changes in value which attract these limits.

In normal PEST practice UPVECBEND should be set to zero or omitted from the PEST control file altogether. Either option deactivates parameter upgrade bending.

4.2.8 Seventh Line

PHIREDSWH

If the parameter-group-specific FORCEN variable appearing in the "parameter groups" section of the PEST control file is set to "switch" or "switch_5", then PEST will switch from single to three-point or five-point derivatives calculation if the inversion process appears to be slowing. Switching takes place if the relative reduction in the objective function between successive iterations is less than the user-supplied value for PHIREDSWH. Thus if, for the *i'*th iteration

$$(\Phi_{i-1} - \Phi_i)/\Phi_{i-1} \leq \text{PHIREDSWH} \qquad (4.2.7)$$

(where Φ_i is the objective function calculated on the basis of the upgraded parameter set determined in the *i'*th iteration), then PEST will use three-point or five-point derivatives in iteration i+1 (and all succeeding iterations) for all parameter groups for which FORCEN is set to "switch" or "switch_5" respectively.

A value of 0.1 is often suitable for PHIREDSWH. If it is set too high, PEST may make the switch to higher order derivatives calculation before it needs to; the result will be that more

model runs will be required to fill the Jacobian matrix than are really needed at that stage of the inversion process. If PHIREDSWH is set too low, PEST may waste an iteration or two in lowering the objective function to a smaller extent than would have been possible if it had made an earlier switch to higher order derivatives calculation. Note that PHIREDSWH should be set considerably higher than the input variable PHIREDSTP which sets one of the termination criteria on the basis of the relative objective function reduction between iterations.

NOPTSWITCH

The optional NOPTSWITCH variable can be used to delay the onset of higher order derivatives calculation. This can be useful where PEST may prematurely "trip" into higher order derivatives calculation before increased derivatives accuracy is really needed. This can occur, for example, where some parameters need to change a great deal before they can cause a noticeable lowering of the objective function, but are prevented from doing so (or prevent other parameters from doing so) because of the action of the parameter upgrade limiting variables FACPARMAX, RELPARMAX and ABSPARMAX(*N*). In cases like this, model run efficiency is better served if the inversion process maintains two point derivatives calculation until the offending parameter(s) have moved a sufficient distance in parameter space for their effect on the objective function to be noticeable. If this is not done, the premature introduction of higher order derivatives calculation may simply increase the number of model runs required for completion of the inversion process, with no real benefits to this process being accrued.

If supplied, NOPTSWITCH (an integer variable) must be 1 or greater. If it is greater than 1, PEST will not switch to higher order derivatives calculation until the NOPTSWITCH'th iteration at least, as long as the objective function does not rise during any iteration. If the objective function does, in fact, rise, then the NOPTSWITCH setting is overridden and PEST switches to higher order derivatives calculation anyway.

The optimal value for NOPTSWITCH is case-specific; however a value of 3 or 4 often works well. If in doubt omit it, or provide it with a value of 1. If supplied, it must immediately follow PHIREDSWH.

SPLITSWH

SPLITSWH determines when, in the inversion process, split slope analysis begins. Recall from section 3.5.7 of this manual that split slope analysis provides a mechanism for reducing the deleterious impacts of poor model numerical performance on finite-difference derivatives calculation. However, in contrast to NOPTSWITCH, SPLITSWH is a real (rather than integer) variable.

Suppose that SPLITSWH is set to 1.10. Then split slope analysis will begin when the following conditions have been met.

1. Central derivatives are being computed for at least one parameter group. (This is governed by the FORCEN and NOPTSWITCH variables.)
2. At least one iteration has elapsed since switching to the use of central derivatives. (This gives central derivatives without split slope analysis a chance to lower the objective function.)
3. The ratio of the best objective function achieved during a certain iteration to that achieved in the previous iteration is 1.10 or higher.

SPLITSWH can be set to less than 1.0 if desired. For example, if it is set to 0.95, then the use of split slope analysis is triggered when the new-to-old objective function ratio is 0.95 or above. If SPLITSWH is provided with a zero or negative value it is ignored. Hence if split slope analysis is activated, it takes place regardless of the value of the objective function in one iteration relative to that calculated during the previous iteration.

If supplied, SPLITSWH must immediately follow NOPTSWITCH on the seventh line of the "control data" section of the PEST control file. The NOPTSWITCH variable must also therefore be provided. In normal PEST usage there is no need to supply this variable, even if split slope analysis is activated.

DOAUI

DOAI is an optional text variable. If supplied, if follows PHIREDSWH on the seventh line of the "control data" section of the PEST control file; alternatively, if NOPTSWITCH and/or SPLITSWH are supplied, it follows them. It should be provided as either "aui" or "noaui". A value of "noaui" switches on "automatic user intervention"; see section 6.3 of this manual. Omission of this variable, or a value of "noaui", disables automatic user intervention.

Only on very rare occasions is automatic user intervention useful. Insensitivity of one or a number of parameters is accommodated far more gracefully using singular value decomposition or LSQR as a solution device for the inverse problem.

If implemented in a certain way, automatic user intervention can defend the inversion process against threats posed by poor finite-difference derivatives born of model numerical granularity. This mode of implementation is activated by setting "aui" to "auid". See section 6.4 for details.

DOSENREUSE

DOSENREUSE is an optional text variable. If supplied, if follows PHIREDSWH on the seventh line of the "control data" section of the PEST control file; alternatively, if NOPTSWITCH and/or SPLITSWH are supplied, it follows them. Its position is interchangeable with that of the DOAUI variable. A value of "senreuse" activates PEST's "sensitivity reuse" functionality; see section 7 of this manual. Omission of this variable, or providing it with a value of "nosenreuse", disables sensitivity reuse.

If DOSENREUSE is set to "senreuse" then an optional "sensitivity reuse" section can be supplied in the PEST control file. Variables within this section allow a user to govern the way in which sensitivity reuse is implemented. If this section is not provided, then PEST uses default values for variables which govern implementation of sensitivity reuse functionality if DOSENREUSE is set to "senreuse".

If DOSENREUSE is set to "nosenreuse", or is omitted from the seventh line of the PEST control file, then the "sensitivity reuse" section is ignored if it appears in the PEST control file.

On most occasions of PEST usage there is little benefit to be gained from invoking PEST's sensitivity reuse functionality.

BOUNDSCALE

As described by Doherty (2015), ideally all parameters undergoing inversion should be transformed prior to estimation such that their prior variances are all the same, and such that

they show no prior spatial correlation, especially if subspace methods such as singular value decomposition or LSQR are used as solution devices for the inverse problem. The type of transformation which achieves this is known as the Kahunen-Loève transformation. It can be shown that prior parameter transformation of this type promulgates a solution to the inverse problem which is of minimum error variance.

A problem with Kahunen-Loève transformation is that, in most environmental modelling contexts, the appropriate transformation cannot be formulated, or can be formulated only with great difficulty. Furthermore, if properly formulated, Tikhonov regularisation can achieve the same thing. Nevertheless, at the very least, an inversion process is often well-served if parameters are internally normalized by their prior standard deviations. Thus, for parameter p_i PEST should ideally estimate the transformed parameter p_i' given by

$$p_i' = \frac{p_i - \underline{p_i}}{\sigma_i} \tag{4.2.8}$$

where σ_i is the standard deviation of the "natural variability" of parameter p_i and $\underline{p_i}$ is its preferred value based on expert knowledge, this presumably being its initial value.

Prior parameter standard deviations are not supplied in a PEST control file. However on many occasions of PEST usage, parameter bounds are a reflection of what a user considers to be prior parameter variability. For example, a user may supply bounds to parameters such that the difference between upper and lower bounds (taking log transformation into account) can be considered as being roughly equivalent to four to six prior standard deviations, this spanning their 95 percent to 99.5 percent prior confidence intervals if parameters are normally distributed.

If parameter bounds can, in fact, be construed as reflective of prior parameter variability then PEST can use these bounds as a basis for internal parameter scaling. This can be implemented by setting the optional BOUNDSCALE text variable to "boundscale". Alternatively, if the BOUNDSCALE variable is omitted, or if it is set to "noboundscale", then internal scaling of parameters does not occur.

With BOUNDSCALE set to "boundscale", internal parameter scaling is implemented under the following circumstances:

- when using singular value decomposition to solve the inverse problem;
- when using LSQR to solve the inverse problem;
- in defining super parameters as part of SVDA-based parameter estimation.

Note that where Tikhonov regularisation is employed, PEST adjusts its application of regularisation to accord with internal parameter transformations effected through bounds scaling. This, and all other aspects of bounds scaling, are transparent to the user.

The following should be noted.

- Obviously, if parameter bounds scaling is activated, then you should try to ensure that parameters bounds are a good reflection of prior parameter uncertainty.
- Scaling of super parameters cannot be implemented. Hence if bounds scaling is activated in a super-parameter PEST control file through which SVD-assisted inversion is undertaken, PEST automatically de-activates it. However PEST does apply bounds scaling to calculation of super parameters from base parameters. This has the same effect as setting the SVDA_SCALADJ control parameter to 4 in the super PEST control file.

- If, for the purposes of SVD-assisted parameter estimation, bounds scaling is activated in the base parameter PEST control file, this will be construed by PEST as an instruction to undertake base parameter bounds scaling in computation of super parameters. This has the same effect as setting the SVDA_SCALADJ control parameter to 4 in the super PEST control file.

(See section 10 of this manual for a description of PEST's SVD-assist methodology.)

If employed, the BOUNDSCALE variable must follow PHIREDSWH and the optional NOPTSWITCH and SPLITSWH variables on the seventh line of the "control data" section of the PEST control file. It can be positioned interchangeably with the DOAUI and DOSENREUSE variables.

Experience has shown that scaling by parameter bounds can, in some circumstances, have a beneficial effect on the efficiency of an inversion process. On other occasions the opposite is the case. On most occasions scaling by bounds is worth a try.

4.2.9 Eighth Line

NOPTMAX

NOPTMAX, an integer variable, sets the maximum number of iterations that PEST is allowed to undertake on a particular parameter estimation run. If you wish to ensure that PEST termination is triggered by other criteria, more indicative of convergence to an optimal parameter, you should set this variable to a high value such as 50.

Two settings for NOPTMAX have special significance however. If NOPTMAX is set to 0, PEST will not estimate parameters, nor even calculate a Jacobian matrix. Instead it will terminate execution after just one model run. This setting can be used to ensure that PEST setup is correct; at the same time, it can be used to compute objective function components and residuals based on initial parameter values provided in the PEST control file.

If NOPTMAX is set to -2 PEST will calculate the Jacobian matrix, store it in a JCO file, and then cease execution immediately. This matrix can then be used for linear analysis, and/or for construction of an SVD-assist input dataset.

Setting NOPTMAX to –1 also instructs PEST to compute the Jacobian matrix. However after doing this, PEST records the same information on its output files as that which it would normally record on completion of an inversion process. This includes composite sensitivities and, if pertinent, post-calibration uncertainty and covariance statistics calculated from parameter sensitivities. It then undertakes a final model run so that model input and output files remaining after PEST execution is complete pertain to parameters values provided in the PEST control file.

PHIREDSTP and NPHISTP

PHIREDSTP is a real variable whereas NPHISTP is an integer variable. If, in the course of the parameter estimation process, there have been NPHISTP optimisation iterations for which

$$(\Phi_i - \Phi_{min})/\Phi_i \leq \text{PHIREDSTP} \qquad (4.2.9)$$

(Φ_i being the objective function value at the end of the i'th optimisation iteration and Φ_{min} being the lowest objective function achieved to date), PEST will consider that the inversion process is at an end.

For many cases 0.005 and 4 are suitable values for PHIREDSTP and NPHISTP respectively. However you must be careful not to set NPHISTP too low if the optimal values for some parameters are near or at their upper or lower bounds (as defined by the parameter variables PARLBND and PARUBND discussed below). In this case it is possible that the magnitude of the parameter upgrade vector may be curtailed over one or a number of iterations to ensure that no parameter value overshoots its bound. The result may be smaller reductions in the objective function than would otherwise occur on these iterations. It would be a shame if these reduced reductions were mistaken for the onset of parameter convergence to the optimal set.

NPHINORED

If PEST has failed to lower the objective function over NPHINORED successive iterations, it will terminate execution. NPHINORED is an integer variable; a value of 3 or 4 is often suitable.

RELPARSTP and NRELPAR

If the magnitude of the maximum relative parameter change between iterations is less than RELPARSTP over NRELPAR successive iterations, PEST will cease execution. The relative parameter change between iterations for any parameter is calculated using equation 4.2.4. PEST evaluates this change for all adjustable parameters at the end of all iterations, and determines the relative parameter change with the highest magnitude. If this maximum relative change is less than RELPARSTP, a counter is advanced by one; if it is greater than RELPARSTP, the counter is zeroed.

All adjustable parameters, whether they are relative-limited or factor-limited, are involved in the calculation of the maximum relative parameter change. RELPARSTP is a real variable for which a value of 0.005 is often suitable. NRELPAR is an integer variable; a value of 3 or 4 is normally satisfactory.

PHISTOPTHRESH, LASTRUN and PHIABANDON

If PHISTOPTHRESH (a real variable) is set to a positive number, PEST will cease execution if the objective function falls below this value at the end of any iteration. Note that this criterion is applied to the measurement objective function if PEST is run in "regularisation" mode (see chapter 9). Alternatively, if PHISTOPTHRESH is set to zero or a negative number, then it is ignored. An error condition will occur if PHISTOPTHRESH is set to a positive number while PEST is asked to run in "predictive analysis" or "pareto" modes.

If supplied, the integer LASTRUN variable must follow the PHISTOPTHRESH variable on the eighth line of the "control data" section of the PEST control file; PHISTOPTHRESH must also then be supplied. If LASTRUN is set to zero, then PEST will not undertake a final model run using optimised parameters upon termination of execution. If it is not set to zero it must be set to one (its default value).

The real-valued PHIABANDON variable optionally follows LASTRUN (which must be provided if PHIABANDON is provided). If, at the end of the first model run, and any iteration thereafter, the objective function (or measurement objective function if PEST is run in "regularisation" mode) is greater than PHIABANDON, PEST will terminate execution. Set this to a very high number, or to a non-positive number (or omit it altogether), for it to have no effect.

Taken together, PHISTOPTHRESH, PHIABANDON and LASTRUN can be useful to control implementation of the null space Monte Carlo methodology wherein a series of randomly-generated, null-space-projected parameter fields are adjusted to respect calibration constraints. Respect for these constraints is deemed to occur when the (measurement) objective function is at or below PHISTOPTHRESH. Once this objective function threshold has been crossed, the optimisation process can be made to cease, without the carrying out of a final model run if LASTRUN is set to 0. PEST can then move on to adjusting another parameter field in the same way (presumably in accordance with commands supplied in a batch or script file). On the other hand if, on undertaking an initial model run when using a new, randomly-generated parameter field, the objective function is so high that it is unlikely to be lowered efficiently to PHISTOPTHRESH, the PHIABANDON variable can instruct PEST to immediately abandon the newly-commenced calibration process in the hope that the next random parameter set offers more potential for recognition of calibration constraints through adjustment of parameter values.

Note that if PEST abandons the parameter estimation process in response to a PHIABANDON setting, it will not undertake a final model run on the basis of "optimised" parameters, regardless of the LASTRUN setting.

If desired, PHIABANDON values can be supplied on an iteration by iteration basis. Thus, for example, you may decide that if the objective function is lower than 10000 when calculated using initial parameters, then it is worth proceeding with the parameter adjustment process. On the other hand, if the objective function has not dropped to 2000 by the end of the second iteration, then the parameter adjustment process should be abandoned. In this case, the name of a file should replace the value of the PHIABANDON variable on the eighth line of the "control data" section of the PEST control file; PEST then reads iteration-specific PHIABANDON values from the cited "PHIABANDON schedule file", an example of which is shown below.

```
10000.0
10000.0
2000.0
```

Figure 4.4 An example if a PHIABANDON schedule file.

Entries in a PHIABANDON schedule file should be listed one to a line. Each represents the PHIABANDON value pertaining to sequential iterations, starting with the zeroth iteration (i.e. the initial model run). The third entry thus pertains to the second iteration. In the example of figure 4.4, if the objective function is above 10000 at the end of the first model run or the end of the first iteration, PEST will abandon the parameter adjustment process. Furthermore if, at the end of the second iteration the objective function is above 2000, it will also take the opportunity to abandon the process.

The following should be noted.

- If there are more entries in the PHIABANDON schedule file than the maximum number of iterations assigned to the current parameter estimation process (i.e. NOPTMAX), PHIABANDON will ignore lines after NOPTMAX+1.
- If there are less than NOPTMAX+1 lines in the PHIABANDON schedule file, PEST will assign to missing iterations the last PHIABANDON value read from the file.
- A zero or negative PHIABANDON value (or a very large value for that matter) can be used to signify that parameter adjustment abandonment is not active during the

pertinent iteration.

4.2.10 Ninth Line

ICOV, ICOR and IEIG

If PEST is run in "estimation" mode, and if neither singular value decomposition nor LSQR is employed in solution of the inverse problem, then at the end of each iteration of the inversion process PEST can write a "matrix file" containing linear estimates of the posterior covariance and correlation coefficient matrices, as well as eigenvectors and eigenvalues of the posterior covariance matrix. Calculation of these matrices is based on the current Jacobian matrix. Settings of the ICOV, ICOR and IEIG variables determine which (if any) of these data are recorded in the matrix file. A setting of 1 for each of these variables will result in the corresponding information being recorded. On the other hand, a setting of 0 will result in the corresponding information not being recorded. Regardless of these settings, all of these matrices are recorded on the run record file at the end of the inversion process (provided, once again, that PEST is not run in "regularisation" mode and that neither singular value decomposition nor LSQR is used to solve the inverse problem.)

The posterior covariance matrix, together with its correlation coefficients and eigenvectors/eigenvalues, can be used to examine the "health" of an inverse problem in which regularisation is purely manual and therefore implied in the definition of the parameters themselves. This covariance matrix is computed using equation 5.2.13 of Doherty (2015). Note, however, that these matrices cannot be computed if an inverse problem is completely ill-posed.

A better way to compute a posterior covariance matrix is to use the PREDUNC7 utility; this is not troubled by problem ill-posedness. Other linear parameter and predictive uncertainty analysis possibilities are available through utilities discussed in part II of this manual. Furthermore, with mathematical regularisation in place, the information provided by eigenvector decomposition of the posterior covariance matrix, and by parameter correlation coefficients computed from this matrix, is not really needed, for the mathematical regularisation process, instead of the user, takes care of possible inverse problem ill-posedness. See Doherty (2015) for details.

IRES

This is an integer variable which controls the writing of a "resolution data file". This is a binary file which is used by utilities such as RESPROC, RESWRIT and PARAMERR which compute the resolution matrix, and quantify post-calibration predictive error variance. The resolution data file is automatically named *case.rsd*, where *case* is the filename base of the PEST control file.

If omitted, IRES is set to 1. Writing of a resolution data file can therefore only be prevented if IRES is included in the PEST control file and specifically set to 0. This is its recommended setting, as the utility programs which use it are less informative than more recent programs such as those comprising the PREDVAR* and PREDUNC* suites which have superseded them; the latter programs do not read a resolution data file.

JCOSAVE

This optional text variable can be used to suppress recording of the binary Jacobian matrix file (i.e. JCO file). If supplied, it must be provided as "jcosave" or "nojcosave". If omitted,

"jcosave" prevails. This variable can be placed anywhere following the mandatory IEIG or optional IRES variable on the ninth line of the "control data" section of the PEST control file. Its recommended value is "jcosave", unless the inverse problem size is very large indeed and the MAXCOMPDIM variable has been employed to activate internal compressed Jacobian matrix storage; in the latter case the accessing of a large Jacobian matrix for the purpose of recording a JCO file can be computationally intensive.

JCOSAVEITN

JCOSAVEITN is an optional character variable which should be supplied as either "jcosaveitn" or "nojcosaveitn". If the former string is supplied, PEST writes a JCO file at the end of every iteration, this containing the Jacobian matrix calculated during that iteration. The name of each such file is *case.jco.N* where *N* is the iteration number to which the JCO file pertains. Alternatively, if JCOSAVEITN is set to "nojcosaveitn", or omitted altogether, PEST will not save a progression of JCO files in this manner. Note that the JCO file containing the Jacobian matrix corresponding to the best parameter set attained so far is saved to a file named *case.jco* in the usual manner, regardless of the setting of JCOSAVEITN.

The JCOSAVEITN variable can be placed anywhere on the ninth line of the "control data" section of the PEST control file following ICOV, ICOR, IEIG and, optionally, IRES.

JCOSAVEITN should be set to "jcosaveitn" if it is desired that the MULJCOSEN utility be employed for monitoring of changes to composite parameter and/or observation sensitivities during the inversion process. See part II of this manual for a description of this utility.

VERBOSEREC

Where parameter, observation and prior information numbers are high, the PEST run record file becomes voluminous - so voluminous that it is almost impossible for a user to obtain useful information from it. PEST will write a much shorter run record file if VERBOSEREC (a text variable) is set to "noverboserec". The default value for this variable (i.e. the value that will prevail if it is not supplied at all) is "verboserec".

The VERBOSEREC variable can be placed anywhere following ICOV, ICOR, IEIG and, optionally, IRES on the ninth line of the "control data" section of the PEST control file.

REISAVEITN

As is documented in the next section, PEST writes an "interim residuals file" at the end of every iteration. This file records observations and corresponding model outputs computed using the best parameters achieved in the inversion process up to that iteration. Regardless of the setting of REISAVEITN, this file is saved (and the previous one overwritten) at the end of every iteration; its name is *case.rei* where *case* is the filename base of the PEST control file.

If the text variable REISAVEITN is set to "reisaveitn", then this same file is also saved as *case.rei.N* where *N* is the current iteration number. Thus at the end of the inversion process, a sequence of files remains in the PEST working directory which collectively document the history of model-to-measurement misfit for every observation comprising the PEST input dataset.

The REISAVEITN variable can be placed anywhere following ICOV, ICOR, IEIG and the optional IRES variable on the ninth line of the "control data" section of the PEST control file.

PARSAVEITN and PARSAVERUN

If supplied, the optional "PARSAVEITN" variable must be recorded as either "parsaveitn" or "noparsaveitn". In the former case PEST will save a parameter value file at the end of every iteration of the inversion process, this containing best parameter values achieved during that iteration, irrespective of whether they improved the overall objective function or not. This file is named *case.par.N* where *case* is the filename base of the PEST control file and *N* is the iteration number. A "parameter history" of the inversion process is thus recorded. Alternatively, if PARSAVEITN is set to "noparsaveitn" or omitted altogether, an iteration-specific parameter value file is not recorded. Neither option interferes with PEST's normal behaviour of recording and refreshing a parameter value file (named *case.par*) containing best parameters achieved to date at the end of every iteration. See chapter 5 for further discussion of this file.

In a similar vein, the variable PARSAVERUN can be used to instruct PEST to record run-specific parameter value files. At the time of writing this functionality is only available if PEST is run as BEOPEST. PARSAVERUN must be set to "parsaverun" or "noparsaverun"; the latter is assumed if it is omitted from the PEST control file.

If PARSAVERUN is set to "parsaverun" then PEST creates a series of parameter value files named *case.par.N_M* where *N* is the run packet index and *M* is the run number within that packet. For identification purposes, run packet indices are also recorded on the run management record file. This file also records run numbers and the nodes to which runs were allocated. Hence a user can associate parameter values with the runs performed by different nodes involved in PEST parallelisation.

Note that if PARSAVERUN is set to "parsaverun" the number of parameter value files that are thereby recorded may be very large indeed.

The PARSAVEITN and PARSAVERUN variables can be placed anywhere following ICOV, ICOR, IEIG and the optional IRES variable on the ninth line of the "control data" section of the PEST control file.

4.3 Sensitivity Reuse Section

PEST's sensitivity reuse functionality, as well as the contents of the "sensitivity reuse" section of the PEST control file, are described in chapter 7 of this manual.

This section is optional. It does not need to be supplied if sensitivity reuse is not implemented. Even if it is implemented, (through setting the DOSENREUSE variable in the "control data" section of the PEST control file to "dosenreuse"), this section can be omitted. Under these circumstances PEST uses default values for all variables which govern implementation of sensitivity reuse functionality.

4.4 Singular Value Decomposition Section

4.4.1 General

Singular value decomposition (i.e. SVD) as a regularisation device, and as a means of introducing numerical stability to solution of an ill-posed inverse problem, is discussed at length in Doherty (2015). The reader is referred to that book for details.

There is really no good reason not to use singular value decomposition as a matter of course

in solving an inverse problem. If a problem is ill-posed then numerical stability of its solution is still guaranteed. Meanwhile, complementary use of Tikhonov regularisation can guarantee sensibility of estimated parameter fields. Hence a "singular value decomposition" section (or an LSQR section) should appear in most, if not all, PEST control files (except when PEST is run in "predictive analysis" mode, where the constrained predictive maximisation/minimisation process that is implemented in this mode is better served by using PEST's default solver).

Variables appearing in the "singular value decomposition" section of the PEST control file are shown in figure 4.5. They are now described in detail.

```
* singular value decomposition
SVDMODE
MAXSING EIGTHRESH
EIGWRITE
```

Figure 4.5 Specifications of the "singular value decomposition" section of the PEST control file.

4.4.2 Second Line

SVDMODE

If SVDMODE is set to zero, use of singular value decomposition is de-activated. Set SVDMODE to 1 or 2 to activate it.

If both singular value decomposition and LSQR are deactivated then PEST solves equation 5.4.2 of Doherty (2015) for parameter upgrades using its default matrix equation solver. If the ($\mathbf{J}^t\mathbf{Q}\mathbf{J} + \lambda\mathbf{I}$) matrix which appears in that equation is not invertible ($\mathbf{J}$ is the Jacobian matrix and $\mathbf{Q}$ is the weight matrix), then this solver can run into numerical quicksand. It can gain some relief through increasing the value of the Marquardt lambda, but this also shortens the length of the parameter upgrade vector, and provides no defence against "hemstitching"; see figure 5.6 of Doherty (2015).

If SVDMODE is set to 1, then PEST uses singular value decomposition to solve this equation. Numerical stability is assured through truncation of singular values before they become very low or zero (see below).

If SVDMODE is set to 2, then PEST undertakes singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix. It then evaluates a parameter upgrade using equation 6.2.4 of Doherty (2015). "Parameter upgrade damping" is implemented through adding a pseudo Marquardt lambda to singular values. Finding the iteration-specific optimal value of this damping parameter is achieved through a trial and error procedure that is exactly the same as that employed for testing different values of the Marquardt lambda.

An SVDMODE setting of 2 is superior to a setting of 1 for cases involving many observations as formulation of the $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix may take a considerable amount of time under these circumstances.

4.4.3 Third Line

MAXSING

MAXSING is the number of singular values before truncation. Ideally this should be equal to the optimal dimensionality of the solution space. This is difficult to determine (though the

SUPCALC utility described in part II of this manual may provide some help in this regard). Furthermore, normally the role of singular value decomposition is to achieve numerical stability of the inversion process rather than to provide regularisation. (Tikhonov constraints normally serve the latter purpose). Hence the EIGTHRESH variable should be used to set the point of singular value truncation. To achieve this, set MAXSING to a number equal to, or greater than, the maximum number of estimable parameters. This is the best setting to use in the vast majority of cases.

EIGTHRESH

EIGTHRESH is the ratio of lowest to highest eigenvalue of the $(\mathbf{J}^t\mathbf{Q}\mathbf{J} + \lambda\mathbf{I})$ matrix at which singular value truncation occurs. For λ equal to zero, this is the square root of the ratio of the highest to lowest singular value of $\mathbf{Q}^{1/2}\mathbf{J}$ at which singular value truncation occurs. To ensure numerical stability of the inversion process this should be set to 5E-7 or above, for this is the value at which numerical noise can be amplified to a level where it smothers the "signal" contained in the calibration dataset. See section 6.2.5 of Doherty (2015) for details. Where numerical malperformance of a model adds considerable numerical noise to finite-difference derivatives, EIGTHRESH may need to be set higher than this – possibly as high as 1E-4.

In the vast majority of cases a setting of 5E-7 is suitable for EIGTHRESH.

4.4.4 Fourth Line

EIGWRITE

When SVD is activated, PEST writes a file named *case.svd* in addition to its usual output files. If SVDMODE is set to 1, this contains singular values (arranged in decreasing order) and corresponding eigenvectors of $(\mathbf{J}^t\mathbf{Q}\mathbf{J} + \lambda\mathbf{I})$ computed on each occasion that singular value decomposition is undertaken. It also records the number of singular values that are actually used in computation of the parameter upgrade vector (i.e. the number of singular values remaining after truncation). Generally, singular value decomposition is carried out at least a number of times per iteration, this corresponding to the testing of different Marquardt lambdas.

If SVDMODE is set to 2, singular values (enhanced by the addition of the pseudo Marquardt lambda) of $\mathbf{Q}^{1/2}\mathbf{J}$ are recorded, together with columns of the $\mathbf{V}_1$ matrix obtained through singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$; see equation 6.2.2 of Doherty (2015).

The SVD output file can become very large; furthermore not all of the information recorded in this file is always worth reading. However sometimes an inspection of singular values can assist in understanding the degree to which an inverse problem is ill-posed. By setting EIGWRITE to 0, only singular values (and not their corresponding eigenvectors), are recorded in *case.svd*, thus reducing its size considerably. The number of singular values used during calculation of each parameter upgrade is also recorded.

4.5 LSQR Section

Once a Jacobian matrix has been calculated and a regularised inverse problem formulated, the LSQR algorithm (Paige and Saunders; 1982a, 1982b) can be used to solve that problem. LSQR can be used interchangeably with singular value decomposition and with PEST's default solver (i.e. the solver that is invoked if neither singular value decomposition nor LSQR is deployed) to solve the linearized inverse problem during each iteration of the overall

inversion process.

The LSQR solution process is similar in many respects to that forthcoming from singular value decomposition. Parameter space is subdivided into estimable and inestimable subspaces. Estimates are calculated only for linear combinations of parameters comprising the former; other combinations remain unchanged from the initial values of these combinations. The process is thus numerically stable, regardless of how ill-posed the inverse problem is. In practice, in order to promulgate not just numerical stability, but also sensibility of estimated parameters, Tikhonov regularisation should also be introduced to an ill-posed inverse problem.

A benefit of LSQR over singular value decomposition is its speed. If parameters number more than a couple of thousand, singular value decomposition can become very slow. This is not the case for LSQR which can happily handle tens of thousands of parameters and hundreds of thousands of observations. The cost of this speed is a slight approximation in separating solution and null subspaces; but this rarely matters.

LSQR can be used for solution of the inverse problem by introducing an "lsqr" section to the PEST control file, and by setting the LSQRMODE variable which appears on the second line of that section to 1. Variables appearing in the LSQR section of the PEST control file are now discussed in detail. Specifications for this file are provided in figure 4.6.

```
* lsqr
LSQRMODE
LSQR_ATOL LSQR_BTOL LSQR_CONLIM LSQR_ITNLIM
LSQRWRITE
```

Figure 4.6 Specifications of the "lsqr" section of the PEST control file.

4.5.1 Second Line

LSQRMODE

LSQRMODE is an integer variable which must be set to 0 or 1. If it is set to 1, LSQR is employed for solution of the inverse problem. The LSQR algorithm is applied to the matrix $\mathbf{Q}^{1/2}\mathbf{J}$ where **J** is the Jacobian matrix and **Q** is the weight matrix.

4.5.2 Third Line

LSQR_ATOL

This is the LSQR *atol* input variable (a real number), described in its documentation as follows. (See below for terminology used in this and other extracts from LSQR documentation.)

> *An estimate of the relative error in the data defining the matrix **A**. For example, if **A** is accurate to about 6 digits, set atol to 1.0e-6.*

The matrix **A** of LSQR documentation is equivalent to the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix used by PEST. Where finite-difference derivatives are used to fill the Jacobian matrix, an ATOL value of 1E-4 generally works well.

LSQR_BTOL

This is the LSQR *btol* input variable (a real number), described in its documentation as follows.

> *An estimate of the relative error in the data defining the rhs vector* ***b****. For example, if* ***b*** *is accurate to about 6 digits, set btol to 1.0e-6.*

A value of 1E-4 generally works well when using PEST.

LSQR_CONLIM

This is the LSQR *conlim* input variable (a real number), described in its documentation as follows.

> *An upper limit on cond(****A****), the apparent condition number of the matrix* ***A****. Iterations will be terminated if a computed estimate of cond(****A****) exceeds conlim. This is intended to prevent certain small or zero singular values of* ***A*** *or* ***A*** *from coming into effect and causing unwanted growth in the computed solution.*
>
> *Conlim and damp may be used separately or together to regularize ill-conditioned systems.*
>
> *Normally, conlim should be in the range 1000 to 1/relpr. Suggested values are:*
>
> *conlim = 1/(100*relpr) for compatible systems,*
>
> *conlim = 1/(10*sqrt(relpr)) for least squares.*
>
> *where relpr is the relative precision of floating-point arithmetic on the machine being used. On most machines, relpr is about 1.0e-7 and 1.0d-16 in single and double precision respectively.*

A value of 1000.0 generally works well with PEST.

LSQR_ITNLIM

This is the LSQR *itnlim* input variable (an integer), described in its documentation as follows.

> *An upper limit on the number of iterations; suggested values are:*
>
> *itnlim = m/2 for well-conditioned systems with clustered singular values;*
>
> *itnlim = 4*m otherwise.*

where m is the number of columns in the matrix **A**.

The number of columns in $\mathbf{Q}^{1/2}\mathbf{J}$ is the number of adjustable parameters. So set LSQR_ITNLIM to four times the number of adjustable parameters.

4.5.3 Fourth Line

LSQRWRITE

If set to 1, output from the LSQR solver will be written to a file named *case.lsq* where *case* is the filename base of the current PEST control file. Information from each call is appended to this file; hence it can become quite lengthy. If this file is not desired, set LSQRWRITE to 0.

4.5.4 Some Notes

A few more words, borrowed from LSQR documentation, serve to explain some of the notation employed above.

LSQR finds a solution **x** *to the following problems:*

1. *Unsymmetric equations - solve* $\mathbf{Ax} = \mathbf{b}$;
2. *Linear least squares - solve* $\mathbf{Ax} = \mathbf{b}$ *in the least-squares sense;*
3. *Damped least squares - solve* $\begin{bmatrix}\mathbf{A}\\ \lambda\mathbf{I}\end{bmatrix}\mathbf{x}=\begin{bmatrix}\mathbf{b}\\ \mathbf{0}\end{bmatrix}$ *in the least-squares sense;*

where $\mathbf{A}$ *is a matrix with n rows and m columns,* $\mathbf{b}$ *is an n-vector, and* λ *is a scalar.* $\underline{\mathbf{A}}$ *and* $\underline{\mathbf{b}}$ *are defined as* $\begin{bmatrix}\mathbf{A}\\ \lambda\mathbf{I}\end{bmatrix}$ *and* $\begin{bmatrix}\mathbf{b}\\ \mathbf{0}\end{bmatrix}$ *respectively.*

In the PEST context, *m* is equivalent to the number of adjustable parameters, and *n* is equivalent to the number of non-zero-weighted observations plus prior information equations.

Note the following.

1. PEST will not allow LSQR to be employed for solution of an inverse problem if automatic user intervention is activated, or if PEST is run in "predictive analysis" mode.
2. PEST permits a "singular value decomposition" section and an "lsqr" section to exist in the same PEST control file. However it does not permit both of SVDMODE and LSQRMODE to have non-zero values; but it does allow both of them to be simultaneously set to zero.
3. In setting values for LSQR control variables, do not forget that the numerical accuracy of terms of the $\mathbf{A}$ matrix is set by the accuracy of derivatives calculation. This is determined by the model if derivatives are supplied externally. However if finite differences are employed for calculation of derivatives, the numerical precision of the elements of $\mathbf{A}$ will probably be of the order of 10^{-2} or 10^{-3} at best.
4. The user is not given the opportunity to supply a value for the LSQR *damp* variable. Rather this is supplied to the LSQR algorithm as implemented in PEST as the current value of the Marquardt lambda. As such, its value is reported to the screen and to the PEST run record file. Variables which control its value are those which govern operation of the Marquardt lambda supplied in the "control data" section of the PEST control file.

4.6 Automatic User Intervention Section

PEST's automatic user intervention functionality, as well as the contents of the "automatic user intervention" section of the PEST control file, are described in section 6.3 of this manual.

This section is optional. It does not need to be supplied if automatic user intervention is not implemented. Even if it is implemented, (through setting the DOAUI variable in the "control data" section of the PEST control file to "doaui"), this section can be omitted. Under these circumstances PEST uses default values for all variables which govern implementation of automatic user intervention.

4.7 SVD Assist Section

As the name suggests, the "svd assist" section of the PEST control file is required for PEST to implement SVD-assisted inversion. The SVD-assist methodology is described in section

10 of this manual, as are the variables which feature in the "svd assist" section of the PEST control file. Normally the PEST control file which is used to implement SVD-assisted inversion is written by the SVDAPREP utility. So it is rare for a user to have to write or edit the contents of the "svd assist" section of a PEST control file him/herself.

4.8 Parameter Groups Section

4.8.1 General

Every parameter must belong to a parameter group; the group to which each parameter belongs is supplied through the parameter-specific PARGP variable supplied in the "parameter data" section of the PEST control file.

Each parameter group must possess a unique name of twelve characters or less in length. However it is strongly recommended that the name be six characters or less in length. As is described in part II of this manual, the ADDREG1 utility (which adds Tikhonov regularisation to a PEST control file) groups prior information equations associated with different parameter groups into different regularisation groups. A prefix of "regul_" is appended to parameter group names in formulating prior information group names. Hence the last six characters of each parameter group are lost in formulating the prior information group name. Uniqueness of prior information group names can then be lost.

Notwithstanding the role of the ADDREG1 utility, the primary purpose of parameter groups is to provide a basis for assignment of variables which govern calculation of finite-difference derivatives. These variables are assigned to parameter groups, rather than to individual parameters, because the number of the latter may be very large in many inversion contexts.

A tied or fixed parameter can be a member of a group; however, as derivatives are not calculated with respect to such parameters, the group to which these parameters belong is of no significance (except, perhaps, in calculating the derivative increment for adjustable group members if the increment type is "rel_to_max" - see below). Hence fixed or tied parameters can be assigned to the dummy group "none" (though this is not recommended practice – see below). If a parameter is assigned to any group other than "none" in the "parameter data" section of the PEST control file, the properties for that group must be defined in the "parameter groups" section of the PEST control file.

The many options available for finite-difference derivatives calculation are discussed in section 3.5 of this manual, as are the variables which control these options. That discussion will not be repeated here. Only a short description of the role of each variable is presented below; the reader is referred to section 3.5 for further details.

Specifications of the "parameter groups" section of the PEST control file are provided in figure 4.7.

```
* parameter groups
PARGPNME INCTYP DERINC DERINCLB FORCEN DERINCMUL DERMTHD [SPLITTHRESH SPLITRELDIFF SPLITACTION]
(one such line for each of NPARGP parameter groups)
```

Figure 4.7 Specifications of the "parameter groups" section of the PEST control file.

4.8.2 Parameter Group Variables

PARGPNME

PARGPNME is the parameter group name. This must be a maximum of twelve characters in

length, though for reasons stated above its length is best limited to six characters.

If a group is featured in the "parameter groups" section of a PEST control file it is not essential that any parameters belong to that group. However if, in the "parameter data" section of the PEST control file, a parameter is declared as belonging to a group that is not featured in the "parameter groups" section of the PEST control file, an error condition will arise.

The parameter group name "none" is illegal. This name is reserved for fixed and tied parameters. However assignment of such parameters to the "none" group is actually not recommended as you may want to untie or unfix them at a later date. It is better to assign them to a parameter group, even if the group-specific variables which govern derivatives calculation have no meaning for these parameters.

INCTYP and DERINC

INCTYP is a character variable which can assume the values "relative", "absolute" or "rel_to_max". If it is "relative", the increment used for forward-difference calculation of derivatives with respect to any parameter belonging to the group is calculated as a fraction of the current value of that parameter; that fraction is provided as the real variable DERINC. However if INCTYP is "absolute" the parameter increment for parameters belonging to the group is fixed, being again provided as the variable DERINC. Alternatively, if INCTYP is "rel_to_max", the increment for any group member is calculated as a fraction of the group member with highest absolute value, that fraction again being DERINC.

Thus, for example, if INCTYP is "relative" and DERINC is 0.01 (a suitable value in many cases), the increment for each group member for each PEST iteration is calculated as 0.01 times the current value of that member. However if INCTYP is "absolute" and DERINC is 0.01, the parameter increment is the same for all members of the group over all iterations, being equal to 0.01. If INCTYP is "rel_to_max" and DERINC is again 0.01, the increment for all group members is the same for any one iteration, being equal to 0.01 times the absolute value of the group member of highest current magnitude; however the increment may vary from iteration to iteration.

If a group contains members which are fixed and/or tied you should note that the values of these parameters are taken into account when calculating parameter increments using the "rel_to_max" option.

For the "relative" and "rel_to_max" options, a DERINC value of between 0.01 and 0.02 is often appropriate. However no suggestion for an appropriate DERINC value can be provided for the "absolute" increment option; the most appropriate increment will depend on parameter magnitudes.

DERINCLB

If a parameter increment is calculated as "relative" or "rel_to_max", it is possible that it may become too low if the parameter becomes very small or, in the case of the "rel_to_max" option, if the magnitude of the largest parameter in the group becomes very small. A parameter increment becomes "too low" if it does not allow reliable derivatives to be calculated with respect to that parameter because of round off errors incurred in the subtraction of nearly equal model-generated observation values.

To circumvent this possibility, an absolute lower bound can be placed on parameter increments; this lower bound is the same for all group members, and is provided as the value

of the DERINCLB variable. Thus if a parameter value is currently 1000.0 and it belongs to a group for which INCTYP is "relative", DERINC is 0.01, and DERINCLB is 15.0, the parameter increment will be 15.0 instead of 10.0 calculated on the basis of DERINC alone. If you do not wish to place a lower bound on parameter increments in this fashion, you should provide DERINCLB with a value of 0.0.

Note that if INCTYP is "absolute", DERINCLB is ignored.

FORCEN

The character variable FORCEN (an abbreviation of "forward/central") determines whether derivatives for group members are calculated using forward differences, one of the variants of the central difference method, one of the variants of the five-point method, or whether different alternatives are used in the course of an optimisation run. It must assume one of the values "always_2", "always_3", "always_5", "switch" or "switch_5".

If FORCEN for a particular group is "always_2", derivatives for all parameters belonging to that group will always be calculated using the forward difference method; filling of the columns of the Jacobian matrix corresponding to members of the group will require as many model runs as there are adjustable parameters in the group. If FORCEN is provided as "always_3", the filling of these same columns will require twice as many model runs as there are parameters within the group; however the derivatives will be calculated with greater numerical precision; this will probably have a beneficial effect on PEST's performance. If FORCEN is set to "always_5" then a five point finite-difference stencil is used for derivatives calculation, this requiring four model runs per parameter.

If FORCEN is set to "switch", derivatives calculation for all adjustable group members begins using the forward difference method, but switches to the three-point method for the remainder of the inversion process on the iteration after the relative objective function reduction between successive PEST iterations is less than PHIREDSWH, a value for which is supplied in the "control data" section of the PEST control file. If FORCEN is set to "switch_5", the switch is made to five-point derivatives calculation instead of three-point derivatives calculation. In either case, the switch to higher order derivatives can be postponed until a certain iteration is reached using the optional NOPTSWITCH variable in the "control data" section of the PEST control file.

Experience has shown that in most instances the most appropriate value for FORCEN is "switch". This allows speed to take precedence over accuracy in the early stages of the inversion process when accuracy is not critical to objective function improvement, and accuracy to take precedence over speed later in the process when realisation of a (normally smaller) objective function improvement requires that derivatives be calculated with as much precision as possible.

DERINCMUL

If derivatives are calculated using one of the three-point methods, the parameter increment is first added to the current parameter value prior to a model run, and then subtracted prior to another model run. For five-point methods two such incremental additions and subtractions are made. In some cases it may be desirable to increase the value of the increment for this process above that used for forward difference derivatives calculation. The real variable DERINCMUL allows you to achieve this. If three-point derivatives calculation is employed, the value of DERINC is multiplied by DERINCMUL; this applies whether DERINC holds the increment factor, as it does for the "relative" or "rel_to_max" increment types, or holds

the parameter increment itself, as it does for the "absolute" increment type. The role of DERINCMUL when using a five-point derivatives stencil is similar; see section 3.5.4 of this manual.

A DERINCMUL value of between 1.0 and 2.0 is usually satisfactory.

DERMTHD

As described in section 3.5, there are three variants of the three-point method of derivatives calculation. If FORCEN for a particular parameter group is "always_3" or "switch", you must inform PEST which three-point method to use. This is accomplished through the character variable DERMTHD which must then be supplied as "parabolic", "best_fit" or "outside_pts". Alternatively, if FORCEN is set to "always_5" or "switch_5" then appropriate values for DERMTHD are "minvar" and "maxprec". If FORCEN is set to "always_2" for a particular group, you must still provide one of these five legal values for DERMTHD; however for such a parameter group, the value of DERMTHD has no bearing on derivatives calculation for its member parameters.

On most occasions of PEST usage FORCEN should be set to "switch" while DERMTHD should be set to "parabolic".

SPLITHRESH, SPLITRELDIFF and SPLITACTION

These optional variables are used to implement "split slope analysis". This methodology for combatting the effects of numerical granularity on finite-difference derivatives, and the roles of these variables in controlling this methodology, are described in section 3.5.7 of this manual. If one of these variables is supplied for a particular parameter group, then all of them must be supplied.

Settings for SPLITTHRESH, SPLITRELDIFF and SPLITACTION of 1.0E-4, 0.5 and "smaller" appear to work well on most occasions.

4.9 Parameter Data Section

4.9.1 General

For every parameter cited in a PEST template file, up to ten pieces of information must be provided in the PEST control file. Conversely, every parameter for which there is information in the PEST control file must be cited at least once in a PEST template file.

The "parameter data" section of the PEST control file is divided into two parts; in the first part a line must appear for each parameter. In the second part, a little extra data is supplied for tied parameters (namely the name of the parameter to which each such tied parameter is linked). If there are no tied parameters, the second part of the "parameter data" section of the PEST control file is omitted.

Each item of parameter data is now discussed in detail. Specifications for the "parameter data" section of the PEST control file are provided in figure 4.8.

```
* parameter data
PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM
(one such line for each of NPAR parameters)
PARNME PARTIED
(one such line for each tied parameter)
```

Figure 4.8 Specifications for the "parameter data" section of the PEST control file.

4.9.2 First Part

PARNME

PARNME is the parameter name. Each parameter name must be unique and of twelve characters or less in length; the name is case insensitive.

PARTRANS

PARTRANS is a character variable which must assume one of four values, these being "none", "log", "fixed" or "tied".

If you wish that a parameter be log-transformed throughout the inversion process, the value "log" must be provided. Logarithmic transformation of some parameters may have a profound effect on the success of the inversion process. If a parameter is log-transformed PEST estimates the log of the parameter rather than the parameter itself. Hence the column of the Jacobian matrix pertaining to that parameter actually contains derivatives with respect to the log of the parameter; likewise, data specific to that parameter in the posterior covariance, correlation coefficient and eigenvector matrices computed by PEST (or the PREDUNC7 utility), pertains to the log of the parameter. However when you supply the parameter initial value (PARVAL1) and its upper and lower bounds (PARUBND and PARLBND), these must pertain to the parameter itself; likewise, at the end of the inversion process, PEST provides the optimised parameter value itself rather than the log of its value.

Experience has shown repeatedly that log transformation of at least some parameters can make the difference between a successful inversion process and an unsuccessful one. This is because, in many cases, the linearity approximation on which each PEST iteration is based holds better when certain parameters are log-transformed. It also follows from the fact that, implied in log-transformation, is a normalization of parameters with respect to their innate variability. The inversion process is thereby able to access, to some extent at least, the benefits which follow from Kahunen-Loève transformation of parameters.

However caution must be exercised when designating parameters as log-transformed. A parameter which can become zero or negative in the course of the inversion process must not be log-transformed; hence if a parameter's lower bound is zero or less, PEST will disallow logarithmic transformation of that parameter. (Note, however, that by using an appropriate SCALE and OFFSET, you can ensure that parameters never become negative. Thus if you are estimating the value for a parameter whose domain, as far as the model is concerned, is the interval [-9.99, 10], you can shift this domain to [0.01, 20] for PEST by designating a SCALE of 1.0 and an OFFSET of -10.0. Similarly, if a parameter's model domain is entirely negative, you can make this domain entirely positive for PEST by supplying a SCALE of -1.0 and an OFFSET of 0.0.)

If a parameter is fixed, taking no part in the inversion process, PARTRANS must be supplied as "fixed". If a parameter is linked to another parameter, this is signified by a PARTRANS value of "tied". In the latter case the parameter plays only a limited role in the inversion process. However the parameter to which the tied parameter is linked (this "parent" parameter must be neither fixed nor tied itself) takes an active part in the inversion process; the tied parameter simply "piggy-backs" on the parent parameter, the value of the tied parameter maintaining at all times the same ratio to the parent parameter as the ratio of their initial values. Note that the parent parameter for each tied parameter must be provided in the second part of the "parameter data" section of the PEST control file.

If a parameter is neither fixed nor tied, and is not log-transformed, the parameter transformation variable PARTRANS must be supplied as "none".

Note that if a particular inversion problem would benefit from a more complex parameter transformation type than logarithmic, this can be accomplished using the parameter preprocessor PAR2PAR; see part II of this manual for details.

PARCHGLIM

This character variable is used to designate whether an adjustable parameter is relative-limited, factor-limited or absolute-limited; see section 3.4.8 of this manual and the discussion of the input variables RELPARMAX, FACPARMAX and ABSPARMAX(*N*) above. PARCHGLIM can be provided with a value of "relative" or "factor" to designate that a parameter is subject to a relative limit or a factor limit respectively. Alternatively, the string "absolute(*N*)" can replace "relative" or "factor" as the value of the PARCHGLIM variable, where *N* is a number between 1 and 10. This specifies that its movement is controlled by the absolute limit provided through the "ABSPARMAX(*N*) = limit" string in the "control data" section of the PEST control file, where the integer *N* links these two components of the change limit specification.

The following aspects of change limit specifications should be noted.

- If a parameter is tied or fixed, its change limit is ignored.
- Parameters that are log-transformed cannot be assigned an absolute change limit; they can only be assigned a factor change limit.
- At the end of each iteration of the inversion process, PEST lists the maximum factor and relative change undergone by any parameter. If any parameter changes are absolute-limited, PEST also lists maximum absolute changes undergone by parameters belonging to absolute change limit groups defined by different values of *N*.

PARVAL1

PARVAL1, a real variable, is a parameter's initial value. For a fixed parameter, this value remains invariant during the inversion process. For a tied parameter, the ratio of PARVAL1 to the parent parameter's PARVAL1 sets the ratio between these two parameters that is maintained throughout the inversion process. For an adjustable parameter PARVAL1 is the parameter's starting value which, together with the starting values of all other adjustable parameters, is successively improved during the inversion process.

Ideally, as explained by Doherty (2015), the initial value of a parameter should be the pre-calibration estimate of the parameter's value based on expert knowledge alone. Through use of the ADDREG1 utility (see part II of this manual), this value can also be specified as the parameter's "preferred value" when implementing Tikhonov regularisation.

Caution should be exercised in choosing an initial parameter value of zero for the following reasons.

- A parameter cannot be subject to relative or factor change limits during the first iteration of the inversion process if its value at the start of that iteration is zero. Furthermore FACORIG cannot be used to modify the action of RELPARMAX and FACPARMAX (the relative- and factor-limiting control variables) if a parameter's original value is zero.

- A relative increment for derivatives calculation cannot be evaluated during the first PEST iteration for a parameter whose initial value is zero. If the parameter belongs to a group for which derivatives are, in fact, specified as "relative", a non-zero DERINCLB variable must be provided for that group.
- If a parameter has an initial value of zero, the parameter can be neither a tied nor a parent parameter as the tied:parent parameter ratio cannot be calculated.

PARLBND and PARUBND

These two real variables represent a parameter's lower and upper bound respectively. For adjustable parameters the initial parameter value (PARVAL1) must lie between these two bounds. However for fixed and tied parameters the values you provide for PARLBND and PARUBND are ignored. (The upper and lower bounds for a tied parameter are determined by the upper and lower bounds of the parameter to which it is tied, and by the ratio between the two.)

PARGP

PARGP is the name of the group to which a parameter belongs. As discussed already, parameter group names must be twelve characters or less in length and are case-insensitive.

As derivatives are not calculated with respect to fixed and tied parameters, PEST provides a dummy group name of "none" to which such tied and fixed parameters can be assigned. Note that it is not obligatory to assign such parameters to this dummy group; they can be assigned to another group if you wish. However, any group other than "none" which is cited in the "parameter data" section of the PEST control file must be properly defined in the "parameter groups" section of this file. (For reasons discussed above, assignment of fixed and tied parameters to the parameter group "none" is not recommended practice.)

SCALE and OFFSET

Just before a parameter value is written to a model input file it is multiplied by the real variable SCALE, after which the real variable OFFSET is added. The use of these two variables allows you to redefine the domain of a parameter. Because they operate on the parameter value "at the last moment" before it is written to the model input file, they take no part in the inversion process; in fact they can "conceal" from PEST the true value of a parameter as seen by the model, PEST estimating, instead, the parameter k_p where

$$k_p = (k_m - o)/s \qquad (4.9.1)$$

Here k_p is the parameter optimised by PEST, k_m is the parameter seen by the model, while s and o are the SCALE and OFFSET for that parameter. If you wish to leave a parameter unaffected by SCALE and OFFSET, set the SCALE to 1.0 and the OFFSET to 0.0.

DERCOM

If the NUMCOM variable in the "control data" section of the PEST control file is set to a number that is greater than 1, then PEST can use different commands to run the model when calculating derivatives with respect to different parameters. The value of the DERCOM variable associated with a particular parameter is the number of that command. Commands are listed in the "model command line" section of the PEST control file. They are numbered in order of appearance, starting at 1.

If the JACFILE variable in the "control data" section of the PEST conrol file is set to 1, then the values for some derivatives can be obtained from an external file. If derivatives with respect to a particular parameter are obtained externally in this manner, set DERCOM for that parameter to zero. See chapter 12 of this manual for further details.

4.9.3 Second Part

The second part of the "parameter data" section of the PEST control file consists of one line for each tied parameter; if there are no tied parameters, the second part of the "parameter data" section must be omitted.

Each line within the second part of the "parameter data" section of the PEST control file consists of two entries. The first is PARNME, the parameter name. This must be the name of a parameter already cited in the first part of the "parameter data" section, and for which the PARTRANS variable has been assigned the value "tied". The second entry on the line, the character variable PARTIED, must hold the name of the parameter to which the first-mentioned parameter is tied, i.e. the "parent parameter" of the first-mentioned parameter. The parent parameter must not be a tied or fixed parameter itself.

Note that PEST allows you to link as many tied parameters as you wish to a single parent parameter. However a tied parameter can, naturally, be linked to only one parent parameter.

4.10 Observation Groups Section

Specifications for the "observation groups" section of a PEST control file are provided in figure 4.9.

```
* observation groups
OBGNME [GTARG] [COVFLE]
(one such line for each of NOBSGP observation groups)
```

Figure 4.9 Specifications of the "observation groups" section of the PEST control file.

In the "observation groups" section of the PEST control file a name is supplied for every observation group. Observation group names must be twelve characters or less in length and are case insensitive. A name assigned to one observation group must not be assigned to any other observation group.

Note that prior information equations are also collected into groups; these groups are also referred to as "observation groups" as they perform an identical role to the groups that represent collected observations. The two are not distinguished in the discussion that follows. All of these groups must be listed in the "observation groups" section of the PEST control file.

Observation group names are written one to a line. NOBSGP such names must be provided, where NOBSGP is listed on the third line of the "control data" section of the PEST control file. If PEST is running in "predictive analysis" mode, one of these group names must be "predict". If it is running in "regularisation" mode at least one of these group names must begin with the "regul" character string.

If a covariance matrix is used for observation weights assignment for a particular observation group, the name of the file holding the covariance matrix for that group must be provided following the group name in the "observation groups" section of the PEST control file. The roles of observation and prior information covariance matrices are discussed in section 3.7 of this manual and in Doherty (2015). The formatting of observation covariance matrix files is

discussed in section 4.19 below. If the name of the file which holds a covariance matrix contains a space (not recommended practice), then the name must be enclosed in quotes.

Optionally, the value of a group-specific target measurement objective function can be specified; this variable is named GTARG. If supplied, GTARG must follow the observation group name; it must precede the name of an optional covariance matrix file. GTARG must not be supplied unless PEST is run in "regularisation" mode; even then, this variable is optional. Its role is discussed in section 9.5 of this manual.

As is discussed below, all observations and prior information equations must be assigned to an observation group. However the same observation group cannot contain both observations and prior information equations. These two different components of a calibration dataset must belong to separate groups (and possibly multiple groups in each case).

4.11 Observation Data Section

The "observation data" section of the PEST control file is particularly simple. Its specifications are shown in figure 4.10.

```
* observation data
OBSNME OBSVAL WEIGHT OBGNME
(one such line for each of NOBS observations)
```

Figure 4.10 Specifications of the "observation data" section of the PEST control file.

For every observation cited in a PEST instruction file there must be one line of data in the "observation data" section of the PEST control file. Conversely, every observation for which data is supplied in the PEST control file must be represented in an instruction file.

Each line within the "observation data" section of the PEST control file must contain four items. Each of these four items is now discussed.

OBSNME

This is a character variable containing the observation name. An observation name must be twenty characters or less in length and contain no spaces. It is case-insensitive. Observation names must be unique.

OBSVAL

OBSVAL, a real variable, is the field or laboratory measurement corresponding to a model-generated observation. It is PEST's role to reduce the difference between this number and the corresponding model-calculated number (the difference is referred to as a "residual") through adjustment of parameter values.

WEIGHT

This is the weight attached to each residual in calculation of the objective function. The manner in which weights are used in the parameter estimation process is discussed in section 3.7 of this manual and in Doherty (2015). Note the following.

- An observation can be effectively removed from the inversion process by providing it with a weight of zero. However PEST (and PESTCHEK) will issue an error message if a negative weight is supplied.
- If a covariance matrix is provided for an observation group in the "observation groups" section of the PEST control file, this overrides weights (including weights of

zero) supplied in the "observation data" section of the PEST control file.

The philosophy of weights assignment is a complex topic. Higher weights should be awarded to data with less measurement noise. However measurement noise is not the only consideration, for account must also be taken of so-called "structural noise" that arises from model inadequacies. Weighting strategies that can accommodate structural noise are addressed by Doherty (2015). Assistance in implementing these strategies is provided by utility programs such as PWTADJ1 discussed in part II of this manual. A procedure that often works well is to assign observations of different types, and/or in different part of a model domain, to different observation groups and then to use utility programs such as PWTADJ1 to ensure that the contribution made to the initial objective function by each group is about the same.

When calibrating a surface water model, or when including contaminant concentrations as a component of the calibration dataset of a groundwater model, it may be necessary for weights to be calculated as functions of the measurements themselves. Thus small flows/concentrations (which are often high in information content) are visible in the inversion process, and are thereby able to influence parameter values achieved through that process. Programs supplied with the Ground and Surface Water Utilities (which are downloadable from the PEST web pages) provide ideas and assistance in designing a suitable weighting strategy.

OBGNME

OBGNME is the name of the observation group to which an observation is assigned. When recording the objective function on the run record file, PEST lists the contribution made to the total objective function by each observation group. It is good practice to assign observations of different types to different observation groups. This gives you the ability to adjust observation weights in a way that ensures that one observation type does not dominate others in the inversion process by virtue of a vastly greater contribution to the objective function.

The observation group name supplied here must be also be listed in the "observation groups" section of the PEST control file. Observation group names must be 12 characters or less in length.

4.12 Derivatives Command Line Section

A "derivatives command line" section does not need to appear in a PEST control file unless the model calculates some or all of the elements of the Jacobian matrix itself; this is expected if the JACFILE variable in the "control data" section of the PEST control file is present and non-zero. The "derivatives command line" section contains the command through which the model is instructed to calculate derivatives, as well as the file from which model-calculated derivatives can be read. See section 12 of this manual for details.

4.13 Model Command Line Section

The "model command line" section of the PEST control file supplies the command which PEST must use to run the model. Optionally, this section may contain multiple commands. This occurs where the NUMCOM variable in the "control data" section of the PEST control file is present and is set to a number greater than 1. The use of multiple model commands can be useful at times, particular when using a surrogate or proxy model for derivatives

calculation, as can be achieved when using PEST's observation re-referencing functionality; see section 12.3 and all of section 14 for further details.

Figure 4.11 shows specifications for the "model command line" section of the PEST control file.

```
* model command line
COMLINE
(one such line for each of NUMCOM command lines)
```

Figure 4.11 Specifications of the "model command line" section of the PEST control file.

The command line may be simply the name of an executable file, or it may be the name of a batch or script file containing a complex sequence of steps. Note that you may include the path name in the model command line which you provide to PEST if you wish. If PEST is to be successful in running the model, then the model executable program must reside in the current directory (i.e. folder), or its full path must be provided, or the PATH environment variable must include the folder in which the executable or batch/script file is situated.

Consider the case of a finite-difference model which calculates the stress field surrounding a tunnel. The input file for this model may be very complicated, involving one or a number of large two or three-dimensional arrays. While parameters can be written to such files using appropriate templates, you may prefer a different approach. Perhaps you wish to estimate rock properties within a small number of zones whose boundaries are known, these zones collectively covering the entire model domain. Furthermore, as is often the case, you may have some preprocessing software which is able to construct the large model arrays from the handful of parameters of interest, namely the elastic properties of the zones into which the model domain has been subdivided. In this case it may be convenient to run the preprocessor prior to running the simulator every time PEST runs the model. This can be accomplished by listing the commands to run both programs in a batch or script file called by PEST as "the model"; hence PEST can now write input files for the preprocessor rather than for the model itself. (Note that programs of the Groundwater Utilities downloadable from the PEST web pages accomplish spatial parameterisation tasks similar to this. More complex spatial parameterisation strategies based on pilot points can be implemented using the PLPROC utility.)

Perhaps the model output file is voluminous; in fact, often models of this kind write their data to binary files rather than ASCII files, relying on the user's postprocessing software to make sense of the abundance of model-generated information that is recorded in this file. You may have a postprocessing program which interpolates model-generated stress data to the locations of your stress sensors. In this case PEST should read the postprocessor output file rather than the model output file.

Hence to use PEST in the parameterisation of the above stress-field model, a suitable model command line may be

```
stress
```

where *stress.bat* is a batch file containing the following sequence of commands.

```
prestress3d
stress3d
postress3d
```

Here PRESTRESS3D and POSTRESS3D are the model pre- and postprocessors respectively; STRESS3D is the stress model itself.

You can get even more complicated than this if you wish. For example, a problem that can arise in working with large numerical models is that they do not always converge to a solution according to the model convergence criteria which you, the user, must supply. The popular United States Geological Survey groundwater model, MODFLOW, requires a variable HCLOSE which determines the precision with which heads are calculated by its matrix solver. Variables such as this should be set small so that heads can be calculated with high precision; the accurate calculation of head derivatives depends on this. However if HCLOSE is set too low the solver may never converge to a point where the maximum head correction between successive solver iterations is less than HCLOSE. If this happens MODFLOW will terminate execution with an error message. Unfortunately, it may be very difficult to predict when this will occur as solver behaviour may be perfect for one set of parameters and unsatisfactory for another. Hence, as PEST continually adjusts parameters for derivatives calculation and parameter upgrades, there is a good chance that, on at least one occasion, there will be a solution failure. When this happens PEST will not find the observations it expects on the model output file and will terminate execution with an appropriate error message (unless the LAMFORGIVE or DERFORGIVE control variables prevent PEST from taking this course of action).

One solution to this problem may be to set HCLOSE high enough such that convergence failure will never occur. However this may result in mediocre PEST performance because of inaccurate derivatives calculation. A better solution would be to recode MODFLOW slightly such that it reads HCLOSE from a tiny file called *hclose.dat*, and such that, if it terminates execution because of solution convergence failure, it does so with a non-zero *errorlevel* setting of, say, 100. (Most compilers allow you to set the *errorlevel* environment variable on program run completion through an appropriate *exit* function call.) Then write two small programs, one named HMUL which reads *hclose.dat*, multiplies HCLOSE by 2 and then rewrites *hclose.dat* with the increased HCLOSE value; the second program, named SETORIG, should write the original, low value of HCLOSE to *hclose.dat*. A suitable model batch file may then be as shown in figure 4.12.

```
@echo off
rem Set hclose to a suitably low value
SETORIG
rem Now run the model
:model
MODFLOW
rem Did MODFLOW converge?
if errorlevel 100 goto adjust
goto end
rem Multiply HCLOSE by 2
:adjust
HMUL
rem Now run model
goto model
:end
```

Figure 4.12 An example of an "intelligent" batch file called by PEST as the model.

(Note that there are alternative, simpler solutions to the MODFLOW convergence problem discussed here. Furthermore, modern versions of MODFLOW, and indeed of many other models, implement adaptive time-stepping schemes which go a long way towards mitigating solver convergence difficulties. The purpose of this example is to demonstrate the type of batch processing that may be useful in a model run by PEST.)

Variations of model batch/script file content are endless. You can call one model followed by another, then by another. The third model may or may not require the outputs of the other two. PEST may read observations from the files generated by all of the models or just from the file(s) generated by the last. Another possibility is that the model batch/script file may call the same numerical simulator a number of times, running it over different historical time periods so that measurements made through all these time periods can be simultaneously used in model calibration.

4.14 Model Input/Output Section

The "model input/output" section of the PEST control file relates PEST template files to model input files and PEST instruction files to model output files. PEST will already have been informed of the respective numbers of these files through the variables NTPLFLE and NINSFLE that reside in the "control data" section of the PEST control file.

Specifications of the "model input/output" section of the PEST control file are provided in figure 4.13.

```
* model input/output
TEMPFLE INFLE
(one such line for each of NTPLFLE template files)
INSFLE OUTFLE
(one such line for each of NINSLFE instruction files)
```

Figure 4.13 Specifications of the "model input/output" section of the PEST control file.

For each template file - model input file pair there should be a line within the "model input/output" section of the PEST control file containing two entries, namely the character variables TEMPFLE and INFLE. The first of these is the name of a PEST template file while the second is the name of the model input file to which the template file is matched. Pathnames should be provided for both the template file and the model input file if they do not reside in the current directory. Construction details for template files are provided in chapter 2 of this manual.

It is possible for a single template file to be linked to more than one model input file. (This may occur if, for example, the same model is being run over more than one historical time period and parameter data for the model resides in a different file from excitation data.) A separate line must be provided for each such pair of files in the "model input/output" section of the PEST control file. A model input file cannot be linked to more than one template file.

As has been explained previously in this manual, a model may have many input files. However PEST only needs to know about those that contain parameters.

The second part of the "model input/output" section of the PEST control file contains instruction file - model output file pairs. There should be one line for each of NINSFLE such pairs, the value of NINSFLE having been provided to PEST in the "control data" section of the PEST control file. Pathnames must be provided for both instruction files and model output files if they do not reside in the current directory. Construction details for instruction files are provided in chapter 2 of this manual.

A single model output file may be read by more than one instruction file; perhaps you wish to extract the values for observations of different types from the model output file using different instruction files. However any particular observation can only ever be referenced once; hence a particular instruction file cannot be matched to more than one model output file.

Where the name of a file featured in the "model input/output" section of the PEST control file contains a space, this name should be enclosed in quotes.

4.15 Prior Information Section

If the value of NPRIOR provided in the "control data" section of the PEST control file is not zero, PEST expects NPRIOR articles of prior information. Prior information can be thought of as observations which pertain directly to parameters themselves. As such, they comprise part of the calibration dataset which, together with observations, assists in the estimation of parameters. An "observation" which comprises a prior information equation can involve more than one parameter. However relationships between parameters that are encapsulated in prior information equations must be linear. If parameter-constraining "observations" pertain to nonlinear relationships between parameters, these relationships must be calculated by the model itself. The PAR2PAR utility can accomplish this task; see part II of this manual.

The manner in which prior information equations are recorded in the "prior information" section of the PEST control file is not unlike that in which you would write an equation on paper yourself; however certain strict protocols must be observed. Refer to figure 4.1 for an instance of a PEST control file containing prior information. (Note that PEST utilities such as the ADDREG1 utility described in part II of this manual, as well as some of the programs comprising the Groundwater Utility suite, add prior information to a PEST control file automatically, this saving you the trouble of having to add it yourself.)

Each item on a prior information line must be separated from its neighbouring items by at least one space. Each new article of prior information must begin on a new line. No prior information line is permitted to exceed 300 characters in length; however a continuation character ("&" followed by a space at the start of a line) allows you to write a lengthy prior information equation over several successive lines.

Prior information lines must adhere to the syntax set out in figure 4.14.

```
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of the NPRIOR articles of prior information)
```

Figure 4.14. The syntax of a prior information line.

Each prior information article must begin with a prior information label; this is the character variable PILBL in figure 4.14. Like observation names, this label must be no more than twenty characters in length, is case insensitive, and must be unique to each prior information article.

Following the prior information label is the prior information equation. To the left of the "=" sign there are one or more combinations of a factor (PIFAC) plus parameter name (PARNME), with a "log" prefix to the parameter name if appropriate. PIFAC and PARNME are separated by a "*" character (which must be separated from PIFAC and PARNME by at least one space) signifying multiplication. All parameters referenced in a prior information equation must be adjustable parameters; i.e. you must not include any fixed or tied parameters in an article of prior information. Furthermore, any particular parameter can be referenced only once in any one prior information equation; however, it can be referenced in more than one equation.

The parameter factor must never be omitted. Suppose, for example, that a prior information equation consists of only a single term, namely that an untransformed, adjustable parameter named "par1" has a preferred value of 2.305, and that you would like PEST to include this

information in the inversion process with a weight of 1.0. If this article of prior information is given the label "pi1", the pertinent prior information line can be written as

```
pi1 1.0 * par1 = 2.305 1.0 pr_info
```

If you had simply written

```
pi1 par1 = 2.305 1.0 pr_info
```

PEST would have objected, complaining of a syntax error.

If a parameter is log-transformed, you must provide prior information pertinent to the log of that parameter, rather than to the parameter itself. Furthermore, the parameter name must be placed in brackets and preceded by "log" (note that there is no space between "log" and the following opening bracket). Thus, in the above example, if parameter "par1" is log-transformed, the prior information article should be rewritten as

```
pi1 1.0 * log(par1) = .362671 1.0  pr_info
```

Note that logs are taken to base 10. Though not illustrated, you will also need to review the weight which you attach to this prior information equation by comparing the extent to which you would permit the log of "par1" to deviate from 0.362671 with the extent to which model-generated observations are permitted to deviate from their measured counterparts.

The left side of a prior information equation can be comprised of the sum and/or difference of a number of factor-parameter pairs of the type already illustrated; these pairs must be separated from each other by a "+" or "-" sign, with a space to either side of the sign. For example

```
pi2 1.0 * par2 + 3.43435 * par4 - 2.389834 * par3 = 1.09e3 3.00 group_pr
```

Prior information equations which include log-transformed parameters must express a relationship between the logs of those parameters. For example if you would like the ratio between the estimated values of parameters "par1" and "par2" to be about 40.0, the prior information equation may be written as

```
pi3 1.0 * log(par1) - 1.0 * log(par2) = 1.60206 2.0  group_pr
```

To the right of the "=" sign of each article of prior information are two real variables and a character variable, namely PIVAL, WEIGHT and OBGNME. The first of these is the "observed value" of the prior information equation. The second is the weight assigned to the article of prior information in the parameter estimation process. This can be zero if you wish (thereby removing the prior information equation from consideration), but must not be negative.

The final item on each line of prior information must be the observation group to which the prior information belongs. Recall that each observation, and each element of prior information, cited in a PEST control file must be assigned to an observation group. In the course of implementing the inversion process, PEST calculates the contribution made to the objective function by each such observation group. The name of any observation group to which an item of prior information is assigned, must also be cited in the "observation groups" section of the PEST control file. As was discussed above, the name of an observation group must be twelve characters or less in length. Optionally, the name of a covariance matrix can be placed alongside that of the observation group in the "observation groups" section of the PEST control file. This replaces prior information weights in computing the component of the objective function associated with that group.

When writing articles of prior information you should note that no two prior information

equations should say the same thing. Thus the following pair of prior information lines is illegal.

```
pi1 2.0 * log(par1) + 2.5 * log(par2) - 3.5 * log(par3) = 1.342 1.00 obgp1
pi2 4.0 * log(par1) + 5.0 * log(par2) - 7.0 * log(par3) = 2.684 1.00 obgp2
```

If you wish to break a single prior information article into more than one line, use the continuation character "&". This must be placed at the beginning of each continuation line, separated from the item which follows it by a space. The line break must be placed between individual items of a prior information article; not within an item. Thus the following lines convey the same information as does the first of the above pair of prior information lines.

```
pi1
& 2.0
& *
& log(par1)
& +
& 2.5
& *
& log(par2)
& -
& 3.5
& *
& log(par3)
& =
& 1.342
& 1.00
& obgp1
```

However the following article of prior information is illegal because of the break between "log" and "par2":

```
pi1 2.0 * log(par1) + 2.5 * log
& (par2) - 3.5 * log(par3) = 1.342 1.00 obgp1
```

4.16 Predictive Analysis Section

If PEST is run in "predictive analysis" mode, (i.e. if the PESTMODE variable residing in the "control data" section of the PEST control file is set to "prediction"), then the PEST control file must have a "predictive analysis" section. Details are provided in chapter 8 of this manual.

4.17 Regularisation Section

If PEST is run in "regularisation" mode, (i.e. if the PESTMODE variable residing in the "control data" section of the PEST control file is set to "regularisation"), then the PEST control file must have a "regularisation" section. Details are provided in chapter 9 of this manual.

4.18 Pareto Section

If PEST is run in "pareto" mode, (i.e. if the PESTMODE variable residing in the "control data" section of the PEST control file is set to "pareto"), then the PEST control file must have a "pareto" section. Details are provided in chapter 13 of this manual.

4.19 Covariance Matrix Files

4.19.1 General

As was discussed in section 4.10, if a covariance matrix is supplied for an observation group (which, of course, includes a prior information group), the name of the file which holds that matrix must be placed alongside the name of the group in the "observation data" section of the PEST control file. Figure 4.15 shows a PEST control file in which two observation groups are endowed with observation covariance matrices. These two matrices are contained within files named *cov1.dat* and *cov2.dat*. Weights assigned to individual observations or prior information items which belong to these groups in the "observation data" or "prior information" sections of the PEST control file are then ignored. Equation 3.7.2 is used to compute the contribution made to the objective function by pertinent observation groups rather than equation 3.7.1. Note however, that if observations are assigned to observation groups whose names begin with "regul" while PEST is running in "regularisation" mode, then an overall multiplier is applied to the covariance matrix in accordance with PEST's implementation of Tikhonov regularisation. See chapter 9 of this manual for details.

```
pcf
* control data
restart estimation
3 10 1 3 3
1 1 single point 1 0 0
5 2 0.3 0.01 10
2 3 0.001 0
0.1
30 0.01 5 5 0.01 5
1 1 1
* parameter groups
ro  relative 0.001 0.0001 switch 2 parabolic
* parameter data
ro1 log factor  4.00 0.1 10000 ro 1 0 1
ro2 log factor  5.00 0.1 10000 ro 1 0 1
ro3 log factor  6.00 0.1 10000 ro 1 0 1
* observation groups
obsgp
obsgp1   cov1.dat
obsgp2   cov2.dat
* observation data
ar1   1.21  1.0 obsgp
ar2   1.51  1.0 obsgp
ar3   2.07  1.0 obsgp
ar4   2.94  1.0 obsgp
ar5   4.15  1.0 obsgp
ar6   5.77  1.0 obsgp
ar7   7.78  1.0 obsgp1
ar8   9.99  1.0 obsgp1
ar9   11.8  1.0 obsgp1
ar10  12.3  1.0 obsgp1
* model command line
model.bat
* model input/output
ves.tpl   model.in1
ves.ins   model.out
* prior information
pi1 1.0 * log(ro1) = 1.32 1.0 obsgp2
pi2 1.0 * log(ro2) = 0.45 1.0 obsgp2
pi3 1.0 * log(ro3) = 0.89 1.0 obsgp2
```

Figure 4.15 A PEST control file citing two covariance matrices.

The following rules must be followed when supplying covariance matrices for observation groups.

- The matrix must be square, symmetric and positive definite. (Actually a matrix cannot be positive definite unless it is symmetric).
- The number of rows and columns which comprise the matrix must be the same as the number of observations or prior information equations which comprise the observation group to which the covariance matrix is assigned.

If desired, the same covariance matrix can be provided to multiple observation groups.

4.19.2 Format of a Covariance Matrix File

PEST provides a number of format options for a covariance matrix file. All options require that the matrix be supplied in an ASCII (i.e. text) file; a binary file is not allowed. The options differ in whether a header precedes the matrix, and whether entity names follow the matrix. All of these different options are supported so that PEST can accept matrices written by various utilities supplied with PEST (including the Groundwater Utilities and PLPROC) that follow slightly different protocols for matrix file storage. For all of these options, the following specifications pertain to the matrix itself.

- Elements within the matrix must be space or comma delimited. They are read using free-field formatting.
- Entries for a new row of the matrix must start on a new line.

The simplest option is to supply only the matrix itself. Figure 4.16a shows an example of a small covariance matrix file which follows this protocol.

```
1.0  0.1  0.0  0.0
0.1  1.0  0.1  0.0
0.0  0.1  1.0  0.1
0.0  0.0  0.1  1.0
```

Figure 4.16a. The simplest protocol for a covariance matrix file.

In order to conform with matrix protocols employed by the PLPROC utility, a covariance matrix can be preceded by a header. This header must contain three integers. The first and second are the number of rows and columns pertaining to the matrix. (These must be equal for a covariance matrix.) The third entry must be the integer "1" or the integer "-1". These three integers must be space delimited. See figure 4.16b.

```
4  4  1
1.0  0.1  0.0  0.0
0.1  1.0  0.1  0.0
0.0  0.1  1.0  0.1
0.0  0.0  0.1  1.0
```

Figure 4.16b. The same matrix as in figure 4.16a, but with a header line.

If the third entry on the header line is "-1" instead of "1", then PEST assumes that the file contains only the diagonal elements of the matrix, and that all other elements of the matrix are zero. In that case the matrix itself does not need to be supplied; only its diagonal elements need be supplied. Elements of the one-dimensional vector comprised of the diagonal elements of the matrix must be placed one to a line. Figure 4.16c shows an example.

```
4  4  -1
1.0
2.0
1.0
0.5
```

Figure 4.16c. Protocol for a diagonal matrix.

Optionally the names of elements can follow the matrix in accordance with the "matrix file" protocol adopted by PEST matrix utilities (see part II of this manual). PEST ignores these names. It only ensures that any information that follows the matrix in a matrix file is preceded by a "* row and column names" header as depicted in figure 4.16d.

```
4  4  1
1.0  0.1  0.0  0.0
0.1  1.0  0.1  0.0
0.0  0.1  1.0  0.1
0.0  0.0  0.1  1.0
* row and column names
obs1
obs2
obs3
obs4
```

Figure 4.16d. The same matrix as in figure 4.16a, but with a header line leading the matrix and element names following the matrix.

5. Running PEST

5.1 Introduction

5.1.1 General

This chapter describes how PEST is run in order to solve a basic inverse problem, and the types of files that it writes when doing so. The focus is mainly on running PEST in "estimation" mode in a non-parallel environment. More complex modes of PEST behaviour, and parallelisation of model runs, are the subject of later chapters.

5.1.2 Checking PEST's Input Data

PEST's input file requirements have been discussed in detail in previous chapters. Before submitting these files to PEST for a parameter estimation run, you should check that all information contained in them is syntactically consistent and correct. This can be done using the utility programs PESTCHEK, TEMPCHEK and INSCHEK described in part II of this manual.

PEST carries out some checking of its input dataset itself; if there are any syntax errors in any of its input files, or if some of the data elements are of the incorrect type (for example real instead of integer, integer instead of character), PEST will cease execution with an appropriate error message. However PEST does not carry out extensive consistency checks. Hence, unless you carry out input data checking yourself using the utility programs mentioned above, PEST may commence execution on the basis of an erroneous data set. Sometimes the error will be detected and PEST will terminate execution with an error message. In other cases PEST may commence the inversion process, only to terminate execution at some later stage with a run-time error message that may bear little relation to the inconsistency that gave rise to the problem in the first place.

5.1.3 Versions of PEST

Versions of PEST at the time of writing are listed in Table 5.1.

PEST Executable Program	What it Does
PEST	"Vanilla" PEST. This is a 32 bit executable.
I64PEST	A 64 bit version of PEST compiled with the Intel compiler.
PPEST	Parallel PEST. This is a 32 bit executable.
I64PPEST	A 64 bit version of PPEST compiled with the Intel compiler.
BEOPEST32	An advanced version of Parallel PEST that uses TCP/IP for communication between master and slaves. This is a 32 bit executable.
BEOPEST64	An advanced version of Parallel PEST that uses TCP/IP for communication between master and slaves. This is a 64 bit executable.

Table 5.1. Versions of PEST at the time of writing.

All of the versions of PEST listed in table 5.1 run on a WINDOWS operating system. As PEST source code is freely available, users can compile UNIX versions of PEST, Parallel PEST and BEOPEST themselves.

All of the versions of PEST listed in table 5.1 can be used interchangeably. As far as the inversion engine is concerned, they all share the same source code. However they may not all provide identical solutions to the same inverse problem, particularly if that problem is ill-posed and regularisation has not been properly formulated. Different compilers undertake calculations, and perform code optimisations, in different ways. Normally this does not matter. However it may matter a great deal when attempting to invert an ill-conditioned matrix.

Parallelisation of model runs can also make a difference in some circumstances. While the contents of a Jacobian matrix are unaffected by whether model runs undertaken to fill it are conducted in serial or in parallel, the testing of parameter upgrades based on different values of the Marquardt lambda may be affected by parallelisation. The latter is a serial procedure which involves the making of decisions, particularly if parameters hit their bounds. Parallel PEST and BEOPEST attempt "pre-emptive parallelisation" of these model runs in anticipation of certain outcomes. Also, with multiple processors at its disposal, Parallel PEST and BEOPEST may be able to test more Marquardt lambdas than is practical when using serial PEST. While, in the end, all versions of PEST choose a parameter set for use in the next iteration that is "best" in terms of PEST's current mode of operation, the parameter set may differ between serial and parallel versions of PEST because of the larger number of Marquardt lambdas that may be tested in a parallel environment, and because of differences in PEST's response to parameters hitting their bounds between these two environments. The path taken toward an optimal parameter set may therefore be somewhat different.

Doherty (2015) explains that the outcome of an inversion process is not the "correct" parameter set, particularly where data is scarce and the null space of the Jacobian matrix has many dimensions. The best inversion outcome that can be hoped for is the parameter set of minimised error variance. The PEST suite provides linear and nonlinear tools for assessing the post-calibration uncertainty of an estimated parameter set. Normally this uncertainty is high. Small differences between estimates of the parameter set of minimum error variance computed by different versions of PEST should be seen in this light.

5.1.4 Starting PEST

All of the versions of PEST listed in table 5.1 can be run using the command

```
pest case
```

where "pest" is replaced by the name of the appropriate executable listed in the first column of this table, and *case.pst* is the name of the PEST control file. BEOPEST, however, requires a command line switch to inform it of the port that it must use for TCP/IP communication.

5.1.5 Command Line Switches

If used, command line switches follow the name of the PEST control file in the command to run PEST. For example, if PEST execution is initiated with the "/i" switch, then it is run using the command

```
pest case /i
```

In general, only one command line switch can be used at a time, as the consequences of using these switches are normally mutually exclusive. Exceptions are the "/t" switch and the

BEOPEST "/h" switch. Use of the "/h" switch is mandatory when initiating BEOPEST execution. As there is no overlap between the function of this switch and the function of other PEST switches, any of the latter can also be used when starting BEOPEST. If so, they must be placed before the /h switch. Hence this command is legal.

```
beopest64 case /s /h :4004
```

while this command is not:

```
beopest64 case /h :4004 /s
```

The role of each of the PEST command line switches is now discussed in detail.

"/i" Switch

If PEST is started with the "/i" switch, it immediately prompts for the name of a JCO file (i.e. the binary file in which a Jacobian matrix is stored). It uses the Jacobian matrix stored in this file for its first iteration. Hence it does not need to undertake the model runs required for filling of this matrix based on finite parameter differencing. For all subsequent iterations, however, PEST fills the Jacobian matrix in the usual manner.

The "/i" switch can be useful in a number of contexts. For example it allows you to alter PEST settings and start the inversion process again with little computational penalty. More importantly, it facilitates implementation of calibration-constrained Monte Carlo analysis. Using the PREDUNC7 and RANDPAR utilities discussed in part II of this manual, samples can be taken of a linear approximation to a posterior parameter covariance matrix. The values of parameters that constitute these samples can then be adjusted so that the calibration objective function is reduced below a certain threshold. The same Jacobian matrix can be used for the first (and often only) iteration of this re-adjustment procedure for all sampled parameter sets.

If PEST execution is initiated with the "/i" switch then, as soon as it commences execution, it prompts

```
Enter name of JCO file for first iteration sensitivities:
```

to which you should respond with the name of a JCO file produced during (normally the first iteration of) a previous PEST run. The name of this file can be *case.jco* where *case* is the filename base of the current PEST case. However this is a dangerous procedure as the JCO fill will be destroyed and then overwritten on the current PEST run. It is far better for the JCO file to have a different filename base.

The PEST control file used to produce the JCO file to which PEST gains access through use of the "/i" switch must cite the same parameters (with the same log transformation status) in the same order as the current PEST control file. However prior information can be different between the two files. Also weights can be different. If you have any doubts about compatibility of the current PEST control file with the previous Jacobian matrix file use the JCO2JCO utility to create a new JCO file in which compatibility is guaranteed. Compatibility can be checked using the JCOCHEK utility. See part II of this manual.

"/d" and "/s" Switches

The /d and /s switches are both restart switches. The former is used to restart a serial PEST run while the latter is used to restart a parallel PEST (including BEOPEST) run. When PEST is started with either of these switches it immediately reads a series of binary files that were written on a previous PEST run based on the same PEST control file. The filename base of

most of these files is the same as that of the present PEST control file. It is important to note, however, that these restart files are saved only if the RSTFLE variable in the "control data" section of the PEST control file is set to "restart".

Use of either of these switches instructs PEST to re-commence execution at the same place that its previous execution was interrupted. Therefore no model runs are wasted. A different switch is required for the serial and parallel versions of PEST because PEST stores its restart data in different ways in either case. This arises from the different manner in which model runs are managed in both of these cases.

"/r" and "j" Switches

When PEST is restarted using the "/r" switch, it does not resume execution exactly where it left off; rather it re-commences the parameter estimation process at the beginning of the iteration in which it was previously interrupted.

PEST can also be restarted with the "/j" switch, this being an integral part of its user-interaction functionality. In this case it re-commences execution at that point of the inversion process in which it last attempted to calculate parameter upgrades. See section 6.2 for details. (With the introduction of modern regularisation methods, user intervention is rarely, if ever, required.)

Note the following important points pertaining to PEST's restart switches.

1. Parallel PEST and BEOPEST cannot be restarted with the "/d" switch.
2. Non-Parallel PEST cannot be restarted with the "/s" switch.
3. A previously stopped Parallel PEST run can be restarted as non-Parallel PEST, Parallel PEST or BEOPEST using either the "/r" or "/j" switches.
4. A previously stopped non-Parallel PEST run can be restarted as non-Parallel PEST, Parallel PEST or BEOPEST using either the "/r" or "/j" switches.
5. A previously stopped Parallel PEST or BEOPEST run cannot be restarted as non-Parallel PEST with the "/s" switch.
6. A previously stopped non-Parallel PEST or BEOPEST run cannot be restarted as Parallel PEST with the "/d" switch.

You are probably wondering at this stage why, if the "/s" and "/d" switches have the same role, they cannot be used interchangeably between Parallel and non-Parallel PEST. The reason, as stated above, is that run management is different between these two versions of PEST. For Parallel PEST, runs are not necessarily undertaken in sequence. Run results in the restart file are also not stored in sequence. For non-Parallel PEST, the restart file does not hold run results; rather it holds fragments of the Jacobian matrix. This can save considerably on file storage when higher order differences are used for derivatives calculation. Also, this restart methodology is more easily combined with external derivatives calculation – this being available only in non-Parallel versions of PEST at present.

It is freely admitted however that an all-purpose run manager is needed. This, hopefully, will be an outcome of future PEST development. At that stage the "/s" and "/d" switches will be combined.

"/t" Switch

Using the "/t" switch, you can provide a short text string which is then recorded near the top

of the PEST run record file, and which is also written to the screen during each iteration of the inversion process. Where a sequence of PEST runs is being undertaken (for example as part of null space Monte Carlo generation of calibration-constrained random parameter fields), this allows a user to know the position in the sequence of the current PEST run.

The user-supplied text string must follow the "/t" command line switch. Suppose, for example, that the text string associated with a particular PEST run (for which the name of the PEST control file is *case.pst*) is "my text". This can be provided to PEST by running it as follows.

```
pest case /t "my text"
```

The following should be noted.

- The run descriptor text must be surrounded by double quotes.
- It must immediately follow the "/t" switch.
- If a "/t" switch is provided without an ensuing text run descriptor, PEST records an error message and then ceases execution.
- Other switches can appear on the PEST command line together with the "/t" switch and run descriptor text. These can either precede or follow the "/t" switch and associated run descriptor text. However the "/t" switch must not follow the "/h" switch and ancillary information required for the running of BEOPEST, as these must always appear last on the BEOPEST command line.
- The run descriptor supplied for a restarted PEST run can be different from that supplied for the original PEST run. It is run-specific rather than case-specific.
- If the "/t" switch and text run descriptor are omitted from the command line, then no mention is made of the (blank) run descriptor, either on the screen or on the run record file.

The run descriptor is written at the top of the run record file. It is also recorded on the run record file immediately following a re-commencement of the recording of that file following a restart. It is written to the screen at the start of each PEST iteration. See the following example.

```
 OPTIMISATION ITERATION NO.        : 7
    User run reference text        : "My text"
    Model calls so far             : 117
    Current regularisation weight factor                    :   1.8347
    Current value of measurement objective function         :   1.0144
    Current value of regularisation objective function      :   1.4688

    Starting phi for this iteration                         :   5.9583
    Contribution to phi from observation group "obsgp1"     :  0.53622
    Contribution to phi from observation group "obsgp2"     :  0.47816
    Contribution to phi from observation group "regul"      :   4.9440
```

Figure 5.1 Text written to the screen at the start of a new iteration when PEST is run with the "/t" switch.

"/p1" Switch

The "/p1" switch can only be used with Parallel PEST and BEOPEST. It instructs Parallel PEST or BEOPEST to undertake the first model run of the inversion process in parallel with model runs which are used to fill the Jacobian matrix. The first model run is used for calculation of an initial objective function, and for calculation of reference model outputs

used in finite-difference derivatives calculation. Having this run done in parallel with runs that are undertaken with parameters incrementally varied prevents slaves from standing idle while waiting for completion of this initial run.

Parallelisation of model runs is discussed in detail in chapter 11 of this manual.

"/f" Switch

If PEST or BEOPEST is started with the "/f" command line switch, they do not undertake parameter estimation at all. Instead they undertake a series of model runs specifically for the purpose of calculating model outputs using different sets of parameters. These parameter sets must be supplied in a sequence of parameter value files recorded prior to running these programs. These parameter value files can be constructed using PEST-suite utility programs such as RANDPAR, RANDPAR1 and LHS2PEST. As is discussed in part II of this manual, the protocol for parameter value files is such that a single set of parameters is stored in each file. PEST/BEOPEST expects that these files are named as a sequence with a common filename base. Suppose that the filename base is *sample*. Then the parameter value files must be named *sample1.par*, *sample2.par*, *sample3.par*, etc. The sequence does not need to begin at 1; however there must be no gaps in the sequence. The names of the parameters which appear in each of these files must be the same. They must correspond to parameters that appear in the PEST control file on which basis PEST or BEOPEST is run.

Suppose that PEST is started using the command:

```
pest case.pst /f
```

Then upon commencement of execution it reads the PEST control file *case.pst*. It then issues a series of prompts. These prompts, and possible responses to these prompts, are presented below.

```
PEST will run the model repeatedly, using parameter values recorded
in a sequence of parameter value files. It will record all model-calculated
observations in a run results file.

Enter filename base of parameter value files: sample
Enter first index to use: 101
Enter last index to use: 200

Enter name for run results file: record.rrf
```

Responses to the above prompts instruct PEST to read files *sample101.par* to *sample200.par* to obtain 100 sets of parameter values. (If the names of parameters read from these files do not correspond to parameter names recorded in the PEST control file *case.pst*, PEST will cease execution with an appropriate error message.) PEST then carries out model runs based on these parameter values. After each model run has been completed, PEST calculates objective function components, and writes these to the screen and to its run record file; it also records parameter values and model outputs for that run in the run results file. It then initiates the next model run.

Specifications for a run results file are provided in part II of this manual.

Suppose that BEOPEST is started using the command:

```
beopest64 case /f /h :4004
```

Then, in addition to the above prompts, it also asks:

```
Enter parallel run packet size:
```

Suppose that the response to this prompt is "50". Then the BEOPEST master will carry out model runs in packets of 50. Objective function calculations, and the recording of run results in the run results file then takes place on completion of each packet of runs. It is important to ensure that the run packet size is chosen such that it minimizes the chances of slaves being idle; the number of slaves available for carrying out model runs in the above example should thus be 5, 10, 25, 50 or 100. Actually, there is no reason why the run packet size cannot be equal to the total number of parameter sets for which model outputs must be calculated. This is a matter of convenience. However sometimes a useful choice for run packet size is a number which is a small multiple of the number of available slaves. This allows you to view objective functions before the complete set of model runs is finished, thus allowing verification that the model's behaviour is in accordance with your expectations. On the other hand, if model run times are sensitive to parameter values, some slaves may intermittently become idle while waiting for a packet of model runs to finish because of slow model run times on other slaves. Maximum efficiency is thereby gained if the run packet size is equal to the number of runs which must be undertaken.

In the event of model run failure, neither PEST nor BEOPEST cease execution with an error message if started with the "/f" switch. Nor do PEST and BEOPEST try to repeat a failed model run using another slave. Instead both PEST and BEOPEST assume that the model did not like the parameter set with which it was provided. The objective function is recorded as -1.11E35. All model outputs associated with the offending parameter set are recorded in the run results file as -1.11E35. Model run failure is thus easily recognized.

As is documented in part II of this manual, a parameter value file supplies values for the PRECIS and DPOINT variables used by PEST for writing numbers to model input files; it also provides the SCALE and OFFSET for each parameter. PEST and BEOPEST ignore all of these in reading the sequence of parameter value files to which they are directed when run with the "/f" command line option. Instead, values for all of these variables are obtained from the PEST control file. This frees the parameter value generator which writes the parameter value files from needing to know the values of these control variables.

5.2 The PEST Run Record File

5.2.1 An Example

As PEST executes, it writes a detailed record of the parameter estimation process to file named *case.rec*, where *case* is the filename base of the PEST control file to which it is directed through its command line. Where parameters and/or observations are large in number this file can get very long – so long that it is difficult for a user to navigate his/her way through it. It can be made considerably shorter if the VERBOSEREC variable in the "control data" section of the PEST control file is set to "noverboserec".

Figure 5.2 shows such a run record file; the PEST control file corresponding to figure 5.2 is that shown in figure 4.1. Note that this example does not demonstrate a very good fit between measurement and corresponding model outcomes calculated on the basis of the optimised parameter set. This is because it was fabricated to demonstrate a number of aspects of the parameter estimation process that are discussed in the following pages. Note also that PEST was run in "estimation" mode in order to produce the run record illustrated in figure 5.2. The running of PEST in "predictive analysis", "regularisation" or "pareto" modes yields slightly different run record files.

```
                      PEST RUN RECORD: CASE test


PEST Version: 13.6


PEST run mode:-

   Parameter estimation mode


Case dimensions:-

   Number of parameters                          :     5
   Number of adjustable parameters               :     3
   Number of parameter groups                    :     2
   Number of observations                        :    19
   Number of prior estimates                     :     2


Model command line(s):-

   model.bat


Jacobian command line:-

   na


Model interface files:-

   Templates:
      ves1.tpl
      ves2.tpl
   for model input files:
      a_model.in1
      a_model.in2

   (Parameter values written using single precision protocol.)
   (Decimal point always included.)

   Instruction files:
      ves1.ins
      ves2.ins
      ves3.ins
   for reading model output files:
      a_model.ot1
      a_model.ot2
      a_model.ot3


PEST-to-model message file:-

   na


Derivatives calculation:-

Param        Increment   Increment   Increment   Forward or   Multiplier  Method
group        type                    low bound   central      (central)   (central)
ro           relative    1.0000E-02  1.0000E-04    switch       2.000     parabolic
h            relative    1.0000E-02  1.0000E-04    switch       2.000     parabolic


Parameter definitions:-

Name         Trans-            Change       Initial        Lower          Upper
             formation         limit        value          bound          bound
ro1          fixed               na       0.500000          na             na
ro2          log               factor      5.00000       0.100000       10.0000
ro3          tied ro2            na       0.500000          na             na
h1           none              factor      2.00000       5.000000E-02    100.000
h2           log               factor      5.00000       5.000000E-02    100.000
```

Name	Group	Scale	Offset	Model command number
ro1	ro	1.00000	0.00000	1
ro2	ro	1.00000	0.00000	1
ro3	ro	1.00000	0.00000	1
h1	h	1.00000	0.00000	1
h2	h	1.00000	0.00000	1

Prior information:-

Prior info name		Factor		Parameter		Prior information	Weight
	pi1	1.00000	*	h1	=	1.00000	3.000
	pi2	1.00000	*	log[ro2]	+		
		1.00000	*	log[h2]	=	2.60260	2.000

Prior Info Name	Observation Group
pi1	prgp1
pi2	prgp1

Observations:-

Observation name	Observation	Weight	Group
ar1	1.21038	Cov. Mat.	obsgp1
ar2	1.51208	Cov. Mat.	obsgp1
ar3	2.07204	Cov. Mat.	obsgp1
ar4	2.94056	Cov. Mat.	obsgp1
ar5	4.15787	Cov. Mat.	obsgp1
ar6	5.77620	Cov. Mat.	obsgp1
ar7	7.78940	Cov. Mat.	obsgp1
ar8	9.99743	Cov. Mat.	obsgp1
ar9	11.8307	1.000	obsgp2
ar10	12.3194	1.000	obsgp2
ar11	10.6003	1.000	obsgp2
ar12	7.00419	1.000	obsgp2
ar13	3.44391	1.000	obsgp2
ar14	1.58279	1.000	obsgp2
ar15	1.10380	1.000	obsgp2
ar16	1.03086	1.000	obsgp2
ar17	1.01318	1.000	obsgp2
ar18	1.00593	1.000	obsgp2
ar19	1.00272	1.000	obsgp2

Covariance matrices:-

Covariance matrix for observation group "obsgp1" ---->

0.2500	0.1000	0.000	0.000	0.000	0.000	0.000	0.000
0.1000	0.2500	0.1000	0.000	0.000	0.000	0.000	0.000
0.000	0.1000	0.2500	0.1000	0.000	0.000	0.000	0.000
0.000	0.000	0.1000	0.2500	0.1000	0.000	0.000	0.000
0.000	0.000	0.000	0.1000	0.2500	0.1000	0.000	0.000
0.000	0.000	0.000	0.000	0.1000	0.2500	0.1000	0.000
0.000	0.000	0.000	0.000	0.000	0.1000	0.2500	0.1000
0.000	0.000	0.000	0.000	0.000	0.000	0.1000	0.2500

Control settings:-

Initial lambda	: 10.000
Lambda adjustment factor	: iteration-dependent
Value of variable governing adjustment	: -3.0000
Sufficient new/old phi ratio per optimisation iteration	: 0.30000
Limiting relative phi reduction between lambdas	: 3.00000E-02
Maximum trial lambdas per iteration	: 10
Forgive model run failure during lamda testing	: no
Forgive model run failure during Jacobian runs	: no
Perform Broyden's update of Jacobian matrix	: no
Undertake observation re-referencing	: no
Maximum factor parameter change (factor-limited changes)	: 3.0000
Maximum relative parameter change (relative-limited changes)	: na

```
   Fraction of initial parameter values used in computing
   change limit for near-zero parameters                          : 1.00000E-03
   Allow bending of parameter upgrade vector                      : no
   Allow parameters to stick to their bounds                      : no

   Relative phi reduction below which to begin use of
   central derivatives                                            : 0.10000
   Iteration at which to first consider derivatives switch        :   1

   Relative phi reduction indicating convergence                  : 0.50000E-02
   Number of phi values required within this range                :   4
   Maximum number of consecutive failures to lower phi            :   4
   Minimal relative parameter change indicating convergence       : 0.50000E-02
   Number of consecutive iterations with minimal param change     :   4
   Maximum number of optimisation iterations                      :  50

   Attempt automatic user intervention                            : no

   Attempt reuse of parameter sensitivities                       : no

   Scale parameters by their bounds                               : no


File saving options: -

   Save best JCO file                                             : yes
   Save multiple JCO files                                        : no
   Save multiple REI files                                        : no
   Save multiple PAR files                                        : no


                           OPTIMISATION RECORD


INITIAL CONDITIONS:
   Sum of squared weighted residuals (ie phi)                =   780.55
   Contribution to phi from observation group "obsgp1"       =   395.83
   Contribution to phi from observation group "obsgp2"       =   369.92
   Contribution to phi from observation group "prgp1"        =   14.805

      Current parameter values
      ro1             0.500000
      ro2              5.00000
      ro3             0.500000
      h1               2.00000
      h2               5.00000


OPTIMISATION ITERATION NO.        : 1
   Model calls so far             : 1
   Starting phi for this iteration                           :   780.55
   Contribution to phi from observation group "obsgp1"       :   395.83
   Contribution to phi from observation group "obsgp2"       :   369.92
   Contribution to phi from observation group "prgp1"        :   14.805

       Lambda =   10.000     ----->
          Phi =   566.54      (  0.726 of starting phi)

       Lambda =   4.6416     ----->
          Phi =   561.06      (  0.719 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   561.06

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              5.00000
      ro3              1.00000                ro3             0.500000
      h1               1.87089                h1               2.00000
      h2               10.0766                h2               5.00000
   Maximum   factor change:  2.015     ["h2"]
   Maximum relative change:  1.015     ["h2"]


OPTIMISATION ITERATION NO.        : 2
```

```
   Model calls so far             : 6
   Starting phi for this iteration                          :   561.06
   Contribution to phi from observation group "obsgp1"      :   320.20
   Contribution to phi from observation group "obsgp2"      :   232.59
   Contribution to phi from observation group "prgp1"       :   8.2626
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =   2.1544     ----->
          Phi =   385.36      (  0.687 of starting phi)

       Lambda =   1.0772     ----->
          Phi =   359.28      (  0.640 of starting phi)

       Lambda =  0.53861     ----->
          Phi =   328.04      (  0.585 of starting phi)

       Lambda =  0.26930     ----->
          Phi =   298.24      (  0.532 of starting phi)

       Lambda =  0.13465     ----->
          Phi =   275.39      (  0.491 of starting phi)

       Lambda =  6.73261E-02 ----->
          Phi =   260.65      (  0.465 of starting phi)

       Lambda =  3.36630E-02 ----->
          Phi =   252.16      (  0.449 of starting phi)

       Lambda =  1.68315E-02 ----->
          Phi =   247.59      (  0.441 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   247.59

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.881873                h1               1.87089
      h2               30.2297                h2               10.0766
   Maximum   factor change:  3.000     ["h2"]
   Maximum relative change:  2.000     ["h2"]


OPTIMISATION ITERATION NO.        : 3
   Model calls so far             : 17
   Starting phi for this iteration                          :   247.59
   Contribution to phi from observation group "obsgp1"      :   157.57
   Contribution to phi from observation group "obsgp2"      :   89.838
   Contribution to phi from observation group "prgp1"       :  0.18528
   All frozen parameters freed.
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =  8.41576E-03 ----->
          Phi =   61.737      (  0.249 of starting phi)

   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:   61.737

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.293958                h1              0.881873
      h2               35.5231                h2               30.2297
   Maximum   factor change:  3.000     ["h1"]
   Maximum relative change: 0.6667     ["h1"]


OPTIMISATION ITERATION NO.        : 4
   Model calls so far             : 21
   Starting phi for this iteration                          :   61.737
   Contribution to phi from observation group "obsgp1"      :   22.893
   Contribution to phi from observation group "obsgp2"      :   34.347
   Contribution to phi from observation group "prgp1"       :   4.4973
```

```
   All frozen parameters freed.
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =  1.71182E-03 ----->
          Phi =   57.411      (  0.930 of starting phi)

       Lambda =  2.04775E-04 ----->
          Phi =   57.411      (  0.930 of starting phi)

       Lambda =  1.43100E-02 ----->
          Phi =   57.408      (  0.930 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   57.408
   Relative phi reduction between optimisation iterations less than 0.1000
   Switch to central derivatives calculation

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.255235                h1              0.293958
      h2               46.0126                h2               35.5231
   Maximum   factor change:  1.295     ["h2"]
   Maximum relative change: 0.2953     ["h2"]


OPTIMISATION ITERATION NO.        : 5
   Model calls so far             : 27
   Starting phi for this iteration                          :   57.408
   Contribution to phi from observation group "obsgp1"      :   23.085
   Contribution to phi from observation group "obsgp2"      :   29.316
   Contribution to phi from observation group "prgp1"       :   5.0066
   All frozen parameters freed.
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =  1.43100E-02 ----->
          Phi =   56.527      (  0.985 of starting phi)

       Lambda =  3.47417E-03 ----->
          Phi =   56.527      (  0.985 of starting phi)

       Lambda =  5.89421E-02 ----->
          Phi =   56.526      (  0.985 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   56.526

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.263242                h1              0.255235
      h2               42.4120                h2               46.0126
   Maximum   factor change:  1.085     ["h2"]
   Maximum relative change: 7.8253E-02 ["h2"]


OPTIMISATION ITERATION NO.        : 6
   Model calls so far             : 36
   Starting phi for this iteration                          :   56.526
   Contribution to phi from observation group "obsgp1"      :   22.621
   Contribution to phi from observation group "obsgp2"      :   29.018
   Contribution to phi from observation group "prgp1"       :   4.8878
   All frozen parameters freed.
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =  5.89421E-02 ----->
          Phi =   56.526      (  1.000 of starting phi)

       Lambda =  2.29386E-02 ----->
          Phi =   56.526      (  1.000 of starting phi)

       Lambda =  0.15146     ----->
          Phi =   56.526      (  1.000 of starting phi)
```

```
   No more lambdas: phi rising
   Lowest phi this iteration:   56.526

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.263874                h1              0.263242
      h2               42.4996                h2               42.4120
   Maximum   factor change:  1.002     ["h1"]
   Maximum relative change: 2.4025E-03 ["h1"]


OPTIMISATION ITERATION NO.        : 7
   Model calls so far             : 45
   Starting phi for this iteration                          :   56.526
   Contribution to phi from observation group "obsgp1"      :   22.578
   Contribution to phi from observation group "obsgp2"      :   29.068
   Contribution to phi from observation group "prgp1"       :   4.8796
   All frozen parameters freed.
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =  5.89421E-02 ----->
          Phi =   56.525      (  1.000 of starting phi)

       Lambda =  2.29386E-02 ----->
          Phi =   56.525      (  1.000 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   56.525

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.263775                h1              0.263874
      h2               42.4800                h2               42.4996
   Maximum   factor change:  1.000     ["h2"]
   Maximum relative change: 4.6122E-04 ["h2"]


OPTIMISATION ITERATION NO.        : 8
   Model calls so far             : 53
   Starting phi for this iteration                          :   56.525
   Contribution to phi from observation group "obsgp1"      :   22.585
   Contribution to phi from observation group "obsgp2"      :   29.059
   Contribution to phi from observation group "prgp1"       :   4.8809
   All frozen parameters freed.
       param "ro2" frozen: gradient and update vectors out of bounds

       Lambda =  8.92708E-03 ----->
          Phi =   56.525      (  1.000 times starting phi)

       Lambda =  1.85188E-03 ----->
          Phi =   56.525      (  1.000 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   56.525

      Current parameter values                Previous parameter values
      ro1             0.500000                ro1             0.500000
      ro2              10.0000                ro2              10.0000
      ro3              1.00000                ro3              1.00000
      h1              0.263727                h1              0.263775
      h2               42.4732                h2               42.4800
   Maximum   factor change:  1.000     ["h1"]
   Maximum relative change: 1.7950E-04 ["h1"]

   Optimisation complete: the  4 lowest phi's are within a relative distance
                          of eachother of 5.000E-03
   Total model calls:     61

   The model has been run one final time using best parameters.
   Thus all model input files contain best parameter values, and model
```

output files contain model results based on these parameters.

OPTIMISATION RESULTS

Adjustable parameters ----->

Parameter	Estimated value	95% percent confidence limits lower limit	upper limit
ro2	10.0000	7.42181	13.4738
h1	0.263727	0.132222	0.395233
h2	42.4732	23.6218	76.3689

Note: confidence limits provide only an indication of parameter uncertainty.
They rely on a linearity assumption which may not extend as far in
parameter space as the confidence limits themselves - see PEST manual.

Tied parameters ----->

Parameter	Estimated value
ro3	1.00000

Fixed parameters ----->

Parameter	Fixed value
ro1	0.500000

See file test.sen for parameter sensitivities.

Observations ----->

Observation	Measured value	Calculated value	Residual	Weight	Group
ar1	1.21038	1.62662	-0.416243	Cov. Mat.	obsgp1
ar2	1.51208	2.23784	-0.725755	Cov. Mat.	obsgp1
ar3	2.07204	3.01458	-0.942536	Cov. Mat.	obsgp1
ar4	2.94056	3.95376	-1.01320	Cov. Mat.	obsgp1
ar5	4.15787	5.02039	-0.862518	Cov. Mat.	obsgp1
ar6	5.77620	6.14068	-0.364476	Cov. Mat.	obsgp1
ar7	7.78940	7.20847	0.580933	Cov. Mat.	obsgp1
ar8	9.99743	8.10495	1.89248	Cov. Mat.	obsgp1
ar9	11.8307	8.71350	3.11720	1.000	obsgp2
ar10	12.3194	8.89478	3.42462	1.000	obsgp2
ar11	10.6003	8.41699	2.18331	1.000	obsgp2
ar12	7.00419	6.99630	7.894000E-03	1.000	obsgp2
ar13	3.44391	4.74609	-1.30218	1.000	obsgp2
ar14	1.58279	2.59709	-1.01430	1.000	obsgp2
ar15	1.10380	1.44029	-0.336490	1.000	obsgp2
ar16	1.03086	1.10433	-7.347000E-02	1.000	obsgp2
ar17	1.01318	1.03545	-2.226600E-02	1.000	obsgp2
ar18	1.00593	1.01519	-9.263000E-03	1.000	obsgp2
ar19	1.00272	1.00683	-4.115000E-03	1.000	obsgp2

Prior information ----->

Prior information	Provided value	Calculated value	Residual	Weight	Group
pi1	1.00000	0.263727	0.736273	3.000	prgp1
pi2	2.60260	2.62811	-2.551458E-02	2.000	prgp1

See file test.res for more details of residuals in graph-ready format.

See file test.rsr for details of rotated residuals in graph-ready format.

See file test.seo for composite observation sensitivities.

Objective function ----->

Sum of squared weighted residuals (ie phi) = 56.525

```
  Contribution to phi from observation group "obsgp1"       =   22.588
  Contribution to phi from observation group "obsgp2"       =   29.056
  Contribution to phi from observation group "prgp1"        =   4.8815


Correlation Coefficient ----->

  Correlation coefficient                                   =  0.97870


Analysis of residuals ----->

  All residuals (rotated as necessary):-
     Number of residuals with non-zero weight                        =    21
     Mean value of non-zero weighted residuals                       =  1.8798E-02
     Maximum weighted residual [observation "ar10"]                  =   3.425
     Minimum weighted residual [observation "ar7_r"]                 =  -2.467
     Standard variance of weighted residuals                         =   3.140
     Standard error of weighted residuals                            =   1.772

     Note: the above variance was obtained by dividing the objective
     function by the number of system degrees of freedom (ie. number of
     observations with non-zero weight plus number of prior information
     articles with non-zero weight minus the number of adjustable parameters.)
     If the degrees of freedom is negative the divisor becomes
     the number of observations with non-zero weight plus the number of
     prior information items with non-zero weight.

  Rotated residuals for observation group "obsgp1":-
     Number of residuals with non-zero weight                        =     8
     Mean value of non-zero weighted residuals                       = -0.9667
     Maximum weighted residual [observation "ar8_r"]                 =   1.829
     Minimum weighted residual [observation "ar7_r"]                 =  -2.467
     "Variance" of weighted residuals                                =   2.824
     "Standard error" of weighted residuals                          =   1.680

     Note: the above "variance" was obtained by dividing the sum of squared
     residuals by the number of items with non-zero weight.

  Residuals for observation group "obsgp2":-
     Number of residuals with non-zero weight                        =    11
     Mean value of non-zero weighted residuals                       =  0.5428
     Maximum weighted residual [observation "ar10"]                  =   3.425
     Minimum weighted residual [observation "ar13"]                  =  -1.302
     "Variance" of weighted residuals                                =   2.641
     "Standard error" of weighted residuals                          =   1.625

     Note: the above "variance" was obtained by dividing the sum of squared
     residuals by the number of items with non-zero weight.

  Residuals for observation group "prgp1":-
     Number of residuals with non-zero weight                        =     2
     Mean value of non-zero weighted residuals                       =   1.079
     Maximum weighted residual [observation "pi1"]                   =   2.209
     Minimum weighted residual [observation "pi2"]                   = -5.1029E-02
     "Variance" of weighted residuals                                =   2.441
     "Standard error" of weighted residuals                          =   1.562

     Note: the above "variance" was obtained by dividing the sum of squared
     residuals by the number of items with non-zero weight.


K-L information statistics ----->


  AIC   =   28.79352
  AICC  =   31.29352
  BIC   =   32.97161
  KIC   =   32.50485

Parameter covariance matrix ----->

                ro2           h1           h2
ro2           3.7986E-03   2.7161E-03  -5.4116E-03
h1            2.7161E-03   3.9177E-03  -3.9483E-03
```

```
h2            -5.4116E-03  -3.9483E-03   1.4708E-02


Parameter correlation coefficient matrix ----->

                 ro2           h1           h2
ro2             1.000       0.7041      -0.7240
h1             0.7041        1.000      -0.5201
h2            -0.7240      -0.5201        1.000


Normalized eigenvectors of parameter covariance matrix ----->

               Vector_1     Vector_2     Vector_3
ro2             0.8327      -0.4035      -0.3793
h1             -0.5248      -0.7936      -0.3079
h2              0.1768      -0.4554       0.8726


Eigenvalues ----->

               9.3797E-04   3.0330E-03   1.8453E-02
```

Figure 5.2 A PEST run record file. Figure 4.1 shows the corresponding PEST control file.

The various sections of the PEST run record file are now discussed in detail.

5.2.2 Echoing the Input Data Set

PEST commences execution by reading all of its input data. As soon as this is read, it echoes most of this data to the run record file. Hence the first section of this file is simply a restatement of most of the information contained in the PEST control file. Note that the letters "na" stand for "not applicable"; in figure 5.2, "na" is used a number of times to indicate that a particular PEST variable has no effect on the inversion process. Thus, for example, the type of change limit for parameter "ro1" is not applicable because this parameter is fixed.

It is possible that the numbers cited in the run record file for a parameter's initial value and for its upper and lower bounds will be altered slightly from that supplied in the PEST control file. This will only occur if the space occupied by this parameter in a model input file is insufficient to represent any of these numbers to the same degree of precision as that with which they are cited in the PEST control file. PEST then adjusts its internal representation of these numbers such that they are expressed with the same degree of precision as that with which they are written to model input files.

5.2.3 The Parameter Estimation Record

After echoing its input data, PEST calculates the objective function arising out of the initial parameter set; it records this initial objective function value on the run record file together with the initial parameter values themselves. Then it starts the inversion process in earnest, beginning with the first iteration. After calculating the Jacobian matrix PEST attempts objective function improvement using one or more Marquardt lambdas. As it does this, it records the corresponding objective function value, both in absolute terms and as a fraction of the objective function value at the commencement of the iteration.

During the first iteration of figure 5.1, PEST tests two Marquardt lambdas; because the second lambda results in an objective function fall of less than 0.03 (i.e. PHIREDLAM) relative to the first one tested, PEST does not test any further lambdas. Instead it progresses to the next iteration after listing both the updated parameter values as well as those from

which the updated parameter set was calculated, i.e. those at the commencement of the iteration. Note that the only occasion on which the "previous parameter values" recorded at the end of an iteration do not correspond with those determined during the previous iteration is when the switch to three-point or five-point derivatives calculation has just been made and the previous iteration failed to lower the objective function; on such an occasion, PEST adopts as its starting parameters for the new iteration the parameter set resulting in the lowest objective function value achieved so far.

At the end of each iteration PEST records either two or more (depending on its input settings) very important pieces of information; in the case of figure 5.1 it is two. These are the maximum factor parameter change and the maximum relative parameter change. As was discussed in previous chapters, each adjustable parameter must be designated as factor-limited, relative-limited or absolute-limited; in figure 5.1 all adjustable parameters are factor-limited with a factor limit of 3.0. In highly nonlinear inversion contexts, suitable settings for the factor, relative and absolute change limits (i.e. FACPARMAX, RELPARMAX and ABSOLUTE(*N*)) may be important in achieving stability of the inversion process. Note that, along with the values of the maximum factor and relative parameter changes encountered during the iteration, PEST also records the names of the parameters that underwent these changes. In the absence of any regularisation (for example singular value decomposition, LSQR and/or Tikhonov) this information may be crucial in deciding which, if any, parameters should be fixed, or temporarily held at their current values should trouble be encountered in the inversion process. For details of the options available for user-intervention, see chapter 6 of this manual. Note, however, that the numerical regularisation methodologies to which PEST provides access provide a far better means of achieving inversion stability than manual regularisation and its attendant need for user-intervention.

The recording of the maximum factor, relative and (if pertinent) absolute parameter changes at the end of each iteration can also help you to judge whether you have set pertinent change limit variables (i.e. FACPARMAX, RELPARMAX and possibly one or more ABSOLUTE(*N*) variables) wisely. In the present case only the maximum factor change is needed because no parameters are relative-limited or absolute-limited; the maximum relative parameter change is still recorded, however, because one of PEST's termination criteria involves the use of relative parameter changes. Note that had some of the parameters featured in figure 5.1 been relative-limited or absolute-limited, this part of the run record would have been slightly different in that the maximum factor parameter change would have been provided only for factor-limited parameters, the maximum relative parameter change would have been provided only for relative-limited parameters and the maximum absolute change would have been provided only for parameters to which an absolute(*N*) limit applies (for each *N*). However a further line documenting the maximum relative parameter change for all parameters would have been added because of its pertinence to the aforementioned termination criterion.

The PEST run record of figure 5.1 shows that in iteration 2, one of the parameters, namely "h2", incurs the maximum allowed factor change, thus limiting the magnitude of the parameter upgrade vector. In iteration 3, parameter "h1" limits the magnitude of the parameter upgrade vector through incurring the maximum allowed parameter factor change. It is probable that convergence for this case would have been achieved much faster if FACPARMAX in the PEST control file were set higher than 3.0 (a default value of 10.0 is recommended).

At the beginning of the second iteration, parameter "ro2" is at its upper bound. After

calculating the Jacobian matrix and calculating an upgrade vector, PEST notices that parameter "ro2" does not wish to move back into its domain; so it temporarily freezes this parameter at its upper bound and calculates an upgrade vector solely on the basis of the remaining adjustable parameters. PEST notifies the user that movement of parameter "ro2" is thereby temporarily disabled. At the beginning of iteration 3, parameter "ro2" is released again in case, with the upgrading of the other adjustable parameters during the previous iteration, it wants to move back into the internal part of its domain.

In the third iteration only a single Marquardt lambda is tested, the objective function having been lowered to below 0.3 times its starting value for that iteration through the use of this single lambda; 0.3 is the user-supplied value for the PEST control variable PHIRATSUF.

At the end of iteration 4 PEST calculates that the relative reduction in the objective function from that achieved in iteration 3 is less that 0.1; i.e. it is less than the user-supplied value for the PEST control variable PHIREDSWH. Hence, as the input variable FORCEN for at least one parameter group (both groups in the present example) is set to "switch", PEST records the fact that it will be using central differences to calculate derivatives with respect to the members of these groups from now on. However the use of central derivatives does not result in a significant further lowering of the objective function, nor in a dramatic change in parameter values, the objective function having been reduced nearly as far as possible through the use of forward derivatives only. (In other cases, especially those involving a greater number of adjustable parameters than in the above example, the introduction of central derivatives can often get a stalled inversion process moving again.)

The parameter estimation process recorded in figure 5.1 is terminated at the end of iteration 8, after the lowest 4 (i.e. NPHISTP) objective function values are within a relative distance of 0.005 (i.e. PHIREDSTP) of each other. (As stated elsewhere in this manual, it is not important that PEST be allowed to terminate the inversion process itself. If you feel that the objective function is unlikely to fall much further, you can terminate the inversion process any time that you feel like it. This can save many wasted model runs.)

Note that where PEST lists the current objective function value at the start of the inversion process and at the start of each iteration, it also lists the contribution made to the objective function by each observation group (including the observation group "prgp1" comprised solely of prior information). This is valuable information, for if a user notices that one particular group is either dominating the objective function, or is not "seen" as a result of dominance by another contributor, he/she may wish to adjust observation or prior information weights and start the inversion process again. The PWTADJ1 utility documented in part II of this manual can automate the weights-adjustment procedure.

5.2.4 Optimised Parameter Values and Confidence Intervals

After completing the inversion process, PEST prints the outcomes of this process to the third section of the run record file. First it lists estimated parameter values. It does this in three stages; adjustable parameters, then tied parameters and, finally, any fixed parameters.

If no regularisation is used in the inversion process, PEST then calculates 95 percent confidence limits for estimated parameters. However, you should note carefully the following points about confidence limits.

- Confidence limits can only be obtained if a linear approximation to the posterior covariance matrix has been calculated. If, for any reason, it has not been calculated (e.g. because the inverse problem is ill-posed, because singular value decomposition

or LSQR is used to solve the inverse problem, or because Tikhonov regularisation has been employed) then confidence limits will not be provided. To establish parameter and predictive confidence limits under these conditions, utility programs such as GENLINPRED, PREDUNC7, and others that are documented in part II of this manual must be used.

- As noted in the PEST run record file, parameter confidence limits are calculated on the basis of the same linearity assumption as that which was used to derive the equations for parameter improvement that are implemented during each PEST iteration. If the confidence limits are large they will, in all probability, extend further into parameter space than the linearity assumption itself. This will apply especially to logarithmically-transformed parameters for which the confidence intervals cited in the PEST run record are actually the confidence intervals of the logarithms of the parameters as evaluated by PEST from the covariance matrix. If confidence intervals are exaggerated in the logarithmic domain due to a breakdown in the linearity assumption, they will be very much more exaggerated in the domain of non-logarithmically-transformed numbers.
- No account is taken of parameter upper and lower bounds in the calculation of 95 percent confidence intervals. Thus an upper or lower confidence limit can lie well outside a parameter's allowed domain. In figure 5.1, the upper confidence limit for "ro2" lies well above its allowed upper bound as provided by the parameter input variable PARUBND for this parameter. PEST does not truncate confidence intervals at parameter domain boundaries so as not to provide an unduly optimistic impression of post-calibration parameter certainty.
- Calculated parameter confidence intervals are highly dependent on the assumptions underpinning the model. If the definition of estimable parameters is the outcome of a manual regularisation process in which inestimable system property heterogeneity is eliminated in order to achieve problem well-posedness, then the uncertainties ascribed to these parameters may understate the uncertainties associated with system property detail on which important model predictions may depend. This may lead to serious underestimation of predictive uncertainty.

Confidence limits are not provided for tied parameters. The parent parameters of all tied parameters are estimated with the tied parameters "riding on their backs"; hence the confidence intervals for the respective parent parameters reflect their linkages to the tied parameters.

Note that at the end of a PEST run a listing of the estimated parameter values can also be found in the PEST parameter value file. This has the same filename base as the PEST control file, but has an extension of "*.par*".

5.2.5 Observations and Prior Information

After it has recorded the estimated parameter set on the run record file, PEST records the measured observation values, together with their model-generated counterparts calculated on the basis of the estimated parameter set. The differences between the two (i.e. the residuals) are also listed, together with the user-supplied set of observation weights. Where a covariance matrix is used instead of weights for a particular observation group, the string "Cov.Mat." is recorded instead of the weight supplied in the PEST control file (which is meaningless under these circumstances). Following observations, user-supplied and model-optimised prior

information values are listed; a prior information value is the number on the right side of a prior information equation. As was done for observations, residuals and user-supplied weights are tabulated for prior information equations.

5.2.6 Objective Function

Next the objective function is listed, together with the contribution made to the objective function by the different observation groups.

5.2.7 Correlation Coefficient

The correlation coefficient pertaining to the current parameter estimation problem, calculated using equation 5.3.2 of Doherty (2015), is recorded next in the run record file.

5.2.8 Analysis of Residuals

The next section of the run record file lists a number of statistics pertaining to residuals - first to all residuals, and then separately to each observation group (including any observation groups to which prior information was assigned). Ideally, after the parameter estimation process is complete, weighted residuals should have a mean of zero and be randomly distributed. The information contained in this section of the run record file helps to assess whether this is the case. It also allows the user to immediately identify outliers (those observations for which the residuals are unusually high).

In calculating residual statistics, observations with zero weight are ignored.

5.2.9 Kullback-Leibler (K-L) Information Loss Statistics

The AIC, AICC, BIC and KIC information criteria are next recorded. See section 5.5.6 of Doherty (2015) for definition of these criteria and an accompanying discussion. The following should be noted.

1. The above statistics are calculated only if PEST is run in "estimation" mode; they are not calculated if PEST is run in "predictive analysis", "regularisation" or "pareto" modes. Nor are they computed if solution of the inverse problem is achieved using truncated singular value decomposition or LSQR.
2. If a problem is so ill-determined that the covariance matrix cannot be calculated (by virtue of the fact that $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ cannot be inverted), then these statistics will not be calculated either.

The author must confess that in contexts where highly parameterised inversion and concomitant uncertainty analysis can be carried out using efficient, state-of-the-art regularisation methodologies that are described later in this manual, he can see little use for these statistics. They ignore the fact that the world is a complex place and that "maximum likelihood" loses its meaning in the face of this complexity.

In contrast, statistics such as those computed by the PREDVAR1 utility described in part II of this manual recognise the complexity of natural systems. They also recognise the fact that regularisation of one kind or another is required for unique solution of an inverse problem, and that the "cost of uniqueness" is a significant loss of resolution in representing properties of a natural system in a calibrated model. The methodology employed by the PREDVAR* suite of utilities provides a much more coherent approach to estimation of the optimum number of parameters to employ in solving an inverse problem than do K-L statistics. Using singular value decomposition as an exploration methodology, they play loss of system detail

incurred by using too few parameters against amplification of observation or structural noise incurred by using too many.

Actually, these days there is no need for a modeller to pursue a path of parameter parsimony in calibrating an environmental model. Modern regularisation methodologies guarantee a solution of minimum error variance to the inverse problem of model calibration, regardless of its state of ill-posedness. After a model has been calibrated, methodologies such as null space Monte Carlo then allow exploration of the uncertainties of model predictions to which the major contributors may be parameters that cannot be uniquely estimated - parameters that would be omitted from a model altogether if the advice given by K-L statistics were to be followed. After all, it is not the parameters that *can* be estimated which must be represented in a model if that model is to explore predictive uncertainty; it is normally the parameters that *cannot* be uniquely estimated that require representation in a model that is used for that purpose. See Doherty (2015) for a further discussion.

5.2.10 Post-Calibration Parameter Covariance Matrix

When PEST is run in "estimation" or "predictive analysis" modes, and if neither singular value decomposition nor LSQR is employed for solution of the inverse problem, then three matrices are recorded at the end of the PEST run record file. These are the post-calibration parameter covariance matrix, the parameter correlation coefficient matrix, and the matrix of eigenvectors of the parameter covariance matrix (the latter two matrices being derived from the first).

The post-calibration covariance matrix is calculated using equation 5.2.13 of Doherty (2015). As is explained in that text, uncertainties calculated in this way take no account of the manual regularisation that is normally required to formulate a well-posed inverse problem. Thus they take no account of the fact that parameters adjusted though the inversion process may be averaged over a large part of parameter space (thereby eliminating expressions of system property heterogeneity within that space). Nor do they represent parameters that may be tied or fixed because of their inestimability.

The covariance matrix is always a square symmetric matrix with as many rows (and columns) as there are adjustable parameters; hence there is a row (and column) for every parameter featured in the inversion process which is neither fixed nor tied. The order in which the rows (and columns) are arranged in the covariance matrix is the same as the order of occurrence of adjustable parameters in the previous listing of the estimated parameter values. (This is the same as the order of occurrence of adjustable parameters in both the PEST control file and in the first section of the run record file.)

Being a by-product of the inversion process, the elements of the post-calibration covariance matrix pertain to the parameters that PEST actually adjusts; this means that where a parameter is log-transformed, the elements of the covariance matrix associated with that parameter actually pertain to the logarithm (to base 10) of that parameter. Note also that the variances and covariances occupying the elements of the covariance matrix are valid only insofar as the linearity assumption upon which their calculation is based is valid.

The diagonal elements of the covariance matrix are the post-calibration variances of adjustable parameters; for the run record file represented in figure 5.1 these variances pertain, from top left to bottom right, to the parameters log("ro2"), "h1" and log("h2") in that order. The variance of a parameter is the square of its standard deviation.

The off-diagonal elements of the covariance matrix represent the covariances between

parameter pairs; thus, for example, the element occupying the second row and third column of the above covariance matrix represents the covariance of "h1" with log("h2").

If there are more than eight adjustable parameters, the rows of the covariance matrix are written in "wrap" form; i.e. after eight numbers have been written, PEST commences a new line to write the ninth number. Similarly, if there are more than sixteen adjustable parameters, the seventeenth number begins on a new line. Note, however, that every new row of the covariance matrix begins on a new line of the run record file.

5.2.11 Correlation Coefficient Matrix

The correlation coefficient matrix is calculated from the post-calibration covariance matrix using equation 3.5.3 of Doherty (2015). The correlation coefficient matrix has the same number of rows and columns as does the covariance matrix; furthermore the manner in which these rows and columns are related to adjustable parameters (or their logs) is identical to that of the covariance matrix. Like the covariance matrix, the correlation coefficient matrix is symmetric.

The diagonal elements of the correlation coefficient matrix are always unity; off-diagonal elements are always between 1.0 and -1.0. The closer that an off-diagonal element is to 1.0 or -1.0, the more highly correlated are the parameters corresponding to the row and column numbers of that element. Thus, for the correlation coefficient matrix recorded in the run record file of figure 5.1, the logs of parameters "ro2" and "h2" exhibit a moderate amount of correlation. However none of the correlation coefficients appearing in this covariance matrix are high enough to suggest nonuniqueness of estimability of the pertinent parameters.

As is pointed out by Doherty (2015), while the information contained in the correlation coefficient matrix has the potential to inform a user whether the inverse problem is currently ill-posed (or is approaching this condition), and whether excessive parameter correlation is responsible for this condition, in practice, values calculated for elements of this matrix can often be deleteriously affected by the very ill-posedness that they seek to expose. The eigenvalues and eigenvectors of the post-calibration covariance matrix are a far more reliable source of information in this regard.

5.2.12 Normalised Eigenvector Matrix and Eigenvalues

The eigenvector matrix is composed of as many columns as there are adjustable parameters, each column containing a normalised eigenvector of the post-calibration covariance matrix. Because a covariance matrix is positive definite, these eigenvectors are real and orthogonal; they represent the directions of the axes of the post-calibration probability "ellipsoid" in the *m*-dimensional space occupied by the *m* adjustable parameters.

In the eigenvector matrix the eigenvectors are arranged from left to right in increasing order of their respective eigenvalues; the eigenvalues are listed beneath the eigenvector matrix. The square root of each eigenvalue is the length of the corresponding semi axis of the post-calibration probability ellipsoid in *m*-dimensional adjustable parameter space.

If the ratio of a particular eigenvalue to the lowest eigenvalue of the post-calibration covariance matrix is particularly large, then the respective eigenvector defines a direction of relative insensitivity in parameter space. The eigenvector pertaining to the highest eigenvalue is particularly worthy of attention in many parameter estimation problems, for this defines the direction of maximum insensitivity, and hence of greatest elongation of the post-calibration probability ellipsoid in adjustable parameter space. If this eigenvector is dominated by a

single element, then the parameter associated with that element may be quite insensitive, the "magnitude of its insensitivity" being defined by the square root of the magnitude of the corresponding eigenvalue. However if this eigenvector contains a number of significant components rather than just one, then this is an indication of insensitivity associated with a *group* of parameters (i.e. parameter correlation). The correlated parameters are those whose eigenvector components are significantly non-zero.

The ratio of the highest to lowest eigenvalue of the post-calibration covariance matrix is of particular importance. The square root of this ratio is related to the "condition number" of the $\mathbf{J}^t\mathbf{QJ}$ matrix that PEST must invert when solving for the parameter upgrade vector - see equations 5.2.11 and 5.2.13 of Doherty (2015). If the condition number of a matrix is too high, then inversion of this matrix becomes numerically difficult (leading to a spurious inverse matrix) or even impossible. In the parameter estimation context this is an outcome of the fact that solution of the inverse problem approaches nonuniqueness as elongation of the post-calibration parameter probability ellipsoid increases to infinity. In general, if the ratio of the highest to lowest eigenvalue of the post-calibration covariance matrix is greater than about 10^7, there is a strong possibility that PEST has had serious difficulties in calculating the parameter upgrade vector because of matrix ill-conditioning born of parameter insensitivity and/or correlation. Unless solution of the inverse problem employs singular value decomposition or LSQR (hopefully with the inclusion of Tikhonov regularisation to guide the inversion process towards sensible parameter values), PEST's performance may have been seriously degraded as a result. This is precisely why adoption of these solution procedures is recommended practice.

5.3 Other PEST Output Files

5.3.1 General

All files written by PEST are tabulated in appendix B. Many of these are binary files that are of little significance to a PEST user, but are highly significant to PEST when attempting a mid-run restart and/or when managing parallel model runs. Other files are recorded by PEST specifically for the benefit of the user. Some contain information that allows you to monitor PEST's performance while it is running. Others contain information that support post-run evaluation of the inversion process. These user-beneficial files are now discussed.

5.3.2 Parameter Value File

At the end of each iteration of the inversion process PEST writes the best parameter set achieved so far (this is the set for which the objective function is lowest if PEST is running in "estimation" mode) to a file named *case.par* where *case* is the filename base of the PEST control file; this type of file is referred to as a PEST "parameter value file". At the end of a PEST run, the parameter value file contains the optimal parameter set. Figure 5.3 illustrates such a file. Note that a parameter value file can be used by the TEMPCHEK utility to build a model input file based on a template file, by PESTGEN in assigning initial parameter values to a PEST control file, and by PARREP to build a new PEST control file from an old PEST control file. The RANDPAR utility can generate a suite of parameter value files whose contents constitute samples of a parameter probability density function. See part II of this manual for further details of the way that these, and other PEST utilities, can generate and use parameter value files.

```
single point
```

```
   ro1      1.000000          1.000000         0.0000000
   ro2      40.00090          1.000000         0.0000000
   ro3      1.000000          1.000000         0.0000000
    h1      1.000003          1.000000         0.0000000
    h2      9.999799          1.000000         0.0000000
```

Figure 5.3 A parameter value file.

The first line of a parameter value file cites the character variables PRECIS and DPOINT, the values for which were provided in the "control data" section of the PEST control file. Then follows a line for each parameter. Each line contains a parameter's name, its current value and the values of the SCALE and OFFSET variables for that parameter as supplied in the PEST control file.

Parameter value files named *case.par.N* are saved at the end of every iteration of the inversion process if the PARSAVEITN variable in the "control data" section of the PEST control file is set to "parsaveitn". If the PARSAVERUN variable is set to "parsaverun" and BEOPEST is used, then a series of parameter value files named *case.par.N_M* are recorded, where *N* is the run package index and *M* is the run number. See section 4.2.10 of this manual for details.

5.3.3 Parameter Sensitivity File

The Composite Parameter Sensitivity

Most of the time consumed during each PEST iteration is devoted to calculation of the Jacobian matrix. During this process the model must be run at least *m* times, where *m* is the number of adjustable parameters. As is explained in section 5.3.1 of Doherty (2015), based on the contents of the Jacobian matrix, PEST calculates a figure related to the sensitivity with respect to each parameter of all observations (with the latter weighted as per user-assigned weights). The "composite sensitivity" of parameter *i* is defined as

$$csp_i = \frac{[\mathbf{J}^t\mathbf{Q}\mathbf{J}]_{ii}^{1/2}}{n} \qquad (5.3.1)$$

where **J** is the Jacobian matrix and **Q** is the weight matrix. *n* in equation 5.3.1 is the number of observations with non-zero weights.

Looked at another way, the composite sensitivity of the *i*'th parameter is the normalised (with respect to the number of observations) magnitude of the column of the Jacobian matrix pertaining to that parameter, with each element of that column multiplied by the weight associated with the respective observation. Recall that each column of the Jacobian matrix lists the derivatives of all model-generated observations with respect to a particular parameter.

Immediately after it calculates the Jacobian matrix, PEST writes composite parameter sensitivities to a "parameter sensitivity file" called *case.sen* where *case* is the current case name (i.e. the filename base of the current PEST control file). Figure 5.4 shows an extract from a parameter sensitivity file.

```
PARAMETER SENSITIVITIES: CASE VES4
OPTIMISATION ITERATION NO.  1 ----->
 Parameter_name    Group    Current value      Sensitivity
   ro1             ro          4.00000           1.25387
   ro2             ro          5.00000          0.327518
   ro3             ro          6.00000           2.09172
```

```
  h1              hhh         5.00000          0.176724
  h2              hhh         4.00000        4.718210E-02


OPTIMISATION ITERATION NO.  2 ----->
Parameter_name    Group  Current value        Sensitivity
  ro1             ro          3.79395           1.30721
  ro2             ro          15.0000          0.672146
  ro3             ro          4.57028           1.77164
  h1              hhh         2.85213          0.661729
  h2              hhh         4.00000          0.465682
```

Figure 5.4 Part of a parameter sensitivity file.

The composite sensitivities recorded in the parameter sensitivity file are sensitivities "as PEST sees them". Thus if a parameter is log-transformed, sensitivity is expressed with respect to the log of that parameter.

Where no form of regularisation features in the inverse problem (for example if Tikhonov regularisation is not employed, and/or if neither singular value decomposition nor LSQR is deployed as a solution methodology), then composite parameter sensitivities can be useful in identifying those parameters which may be degrading the performance of the parameter estimation process through lack of sensitivity to model outcomes.

Information is appended to the parameter sensitivity file during each iteration of the inversion process immediately following calculation of the Jacobian matrix. In the event of a restart, the parameter sensitivity file is not overwritten; rather PEST preserves the contents of the file, appending information pertaining to subsequent iterations to the end of the existing file. In this manner the user is able to track variations in the composite sensitivity of each parameter through the parameter estimation process.

When inspecting the parameter sensitivity file, keep the following points in mind.

- If PEST is working in "predictive analysis" mode, it assumes that the weight assigned to the observation constituting the sole member of the observation group "predict" is zero. Thus there is no contribution to the composite sensitivity of any parameter from the sole member of this observation group. However the situation is slightly different for information written to the parameter sensitivity file at the end of the simulation - see below.
- If PEST is working in "regularisation" mode, the weights assigned to members of the observation group "regul" vary from iteration to iteration (see chapter 9). Composite parameter sensitivities for any iteration are calculated using the optimal weight factor (calculated on an iteration-by-iteration basis by PEST) for members of observation groups whose names begin with "regul".
- If an observation covariance matrix is supplied instead of observation weights for any observation group, this is automatically taken into account when computing composite parameter sensitivities.

Sensitivity Information Recorded on Termination of PEST Execution

At the end of the inversion process (or if PEST is halted prematurely using the "stop with statistics" option – see below), PEST provides a complete listing of composite parameter sensitivities based on the best Jacobian matrix computed during the PEST run. "Best" is defined in terms of PEST's task for that run; this may be to minimise the objective function

("estimation" mode), to maximise/minimise a prediction subject to objective function constraints ("predictive analysis" mode), or to minimise the regularisation component of the objective function subject to constraints imposed on the measurement component of the objective function ("regularisation" mode). "Best" has no meaning when PEST is run in "pareto" mode.

The point within the parameter estimation process where the "best" Jacobian matrix was computed varies from run to run. It may have been computed during the last iteration, or it may have been computed some iterations ago, subsequent attempts to improve the outcome of the inversion process since that iteration having met with no success. Note also that if there was a marginal improvement in the outcome of the inversion process during the final iteration, but not enough to warrant the undertaking of another iteration, then sensitivities will not correspond exactly to estimated parameter values, as PEST does not compute another Jacobian matrix before ceasing execution under these conditions. Nevertheless sensitivities computed by PEST on the basis of the near-optimal parameter values which it uses at the beginning of the last iteration should be a close approximation to those that it would calculate using final parameter estimates. However, if you would like to obtain a Jacobian matrix that is calculated using final parameter values, you can do the following.

1. Use the PARREP utility (see part II of this manual) to build a new PEST control file in which initial parameter values are optimised parameter values.
2. Set NOPTMAX in that file to -1, thus requesting that PEST computes a Jacobian matrix, then computes statistics based on this matrix, and then ceases execution.
3. You may wish to set the FORCEN variable for each parameter group to "always_3" (or even "always_5") in this sensitivity-only run, thus ensuring that PEST calculates derivatives with maximum precision.
4. If working in "regularisation" mode, set the initial weight factor (WFINIT) to the optimal weight factor determined during the previous inversion run (unless the regularisation control variable IREGADJ was set to a non-zero value on the previous PEST run, in which case relativity of inter-regularisation group weights will have been disturbed and this methodology will be inapplicable). However a better idea is to use the SUBREG1 utility to eliminate regularisation from the PEST control file before undertaking the sensitivity-only run. This allows you to draw conclusions regarding the information content of the calibration dataset alone and its ability (or lack thereof) to support a unique solution the inversion process.
5. Then run PEST.

When writing "completion parameter sensitivities" at the end of a parameter sensitivity file, PEST lists the composite sensitivity to each parameter of all observation groups, as well as of each individual observation group. The composite parameter sensitivity of each observation group is evaluated by calculating the magnitude of the respective column of the weighted Jacobian matrix using equation 5.3.1, with the summation confined to members of that particular observation group. The magnitude is then divided by the number of members of that observation group which have non-zero weights.

When PEST is run in "predictive analysis" mode, the observation group "predict" deserves special attention. As was mentioned above, it is not included in the computation of overall parameter sensitivities when PEST is run in this mode. However, because it is a separate observation group, PEST lists the composite sensitivity to each parameter of the single

member of this group, together with composite sensitivities of other observation groups, at the end of the parameter sensitivity file. The observation weight used in this calculation is the weight assigned to the observation comprising the sole member of the observation group "predict" in the PEST control file.

When using PEST in "regularisation" mode, weights assigned to observation groups whose names begin with "regul" are multiplied by the optimal regularisation weight factor determined as part of the inversion process before recording composite sensitivities of parameters to these groups; if IREGADJ is set to a non-zero value, inter-group weighting relatively is also optimised.

5.3.4 Observation Sensitivity File

The composite observation sensitivity of observation o_j is defined as

$$cso_j = \frac{\left[\mathbf{Q}^{1/2}\mathbf{J}\mathbf{J}^t\mathbf{Q}^{1/2}\right]_{jj}^{1/2}}{m} \qquad (5.3.2)$$

That is, the composite sensitivity of observation *j* is the magnitude of the *j'*th row of the Jacobian matrix multiplied by the weight associated with that observation; this magnitude is then divided by the number of adjustable parameters. It is thus a measure of the sensitivity of that observation to all parameters involved in the parameter estimation process; see section 5.3.3 of Doherty (2015) for further details. At the end of its run, PEST lists all observation values and corresponding model-calculated values, as well as composite sensitivities for all observations in an "observation sensitivity file". This file is named *case*.*seo*.

Though composite observation sensitivities can be of some use, they do not, in general, convey as much useful information as do composite parameter sensitivities. In fact in some instances the information that they provide can even be a little deceptive. Thus while a high value of composite observation sensitivity would, at first sight, indicate that an observation is crucial to the inversion process because of its high information content, this may not necessarily be the case. Another observation made at nearly the same time and/or place as the first observation may carry nearly the same information. In this case, it may be possible to omit one of these observations from the inversion process with impunity, for the information which it carries is redundant as long as the other observation is included in the process. Thus while a high value of composite observation sensitivity does indeed mean that the observation to which it pertains is possibly sensitive to many parameters, it does not indicate that the observation is particularly indispensable to the inversion process, for this can only be decided in the context of the presence or absence of other observations with similar sensitivities.

Figure 5.5 shows part of an observation sensitivity file.

```
Observation   Group         Measured          Modelled          Sensitivity
  ar1          group_1        1.210380          1.639640          0.5221959
  ar2          group_1        1.512080          2.254750          0.6824375
  ar3          group_1        2.072040          3.035590          0.8591846
  ar4          group_1        2.940560          3.978450          1.0338167
  ar5          group_1        4.157870          5.047430          1.1915223
  ar6          group_1        5.776200          6.167830          1.3226952
  ar7          group_2        7.789400          7.232960          1.4450249
  ar8          group_2        9.997430          8.124100          1.5881968
  ar9          group_2        11.83070          8.724950          1.7506757
  ar10         group_2        12.31940          8.895600          1.8875951
  ar11         group_2        10.60030          8.402450          1.9690974
```

Figure 5.5 Part of an observation sensitivity file.

5.3.5 Residuals File

On completion of execution, PEST writes a "residuals file" tabulating observation names, the groups to which observations belong, measured and modelled observation values, differences between these two (i.e. residuals), measured and modelled observation values multiplied by respective weights, weighted residuals, measurement standard deviations and "natural weights". This file can be readily imported into a spreadsheet for various types of analysis and plotting. Its name is *case*.*res* where *case* is the current PEST case name.

A word of explanation is required concerning the last two data types presented in the residuals file. On completion of the inversion process, PEST calculates a "reference variance" (also known as "standard variance of weighted residuals"), σ^2_r as

$$\sigma_r^2 = \frac{\Phi_0}{n-m} \tag{5.3.3}$$

where Φ_0 is the objective function corresponding to estimated parameters, n is the number of observations and m is the number of adjustable parameters. Where PEST is run in "estimation" mode, Φ_0 is the minimised objection function. Where PEST is run in "predictive analysis" mode this is the objective function constraint that is operative during the maximisation or minimisation process which PEST undertakes when run in this mode. Where PEST is run in "regularisation" mode Φ_0 includes regularisation observations and prior information. When run in the latter two modes, the reference variance does not have as much meaning as when run in "estimation" mode. See section (5.2.2) of Doherty (2015) for further discussion.

"Natural weights" as represented on the observation residuals file are user-supplied weights divided by the square root of the reference variance. However for observation groups which are actually regularisation groups, user-supplied weights are first adjusted in accordance with PEST's implementation of Tikhonov regularisation.

Where a covariance matrix is supplied for one or more observation groups instead of weights, the residuals file is slightly modified. As in the PEST run record file, weights are not recorded in the residuals file. Instead, where weights should be written, the string "Cov. Mat." is recorded. The string "na" for "not applicable" is recorded where weighted measurements, weighted observations, weighted residuals, measurement standard deviations and natural weights would otherwise be recorded.

Where a covariance matrix is assigned to one or more observation groups, PEST writes a "rotated residuals file" as well as an ordinary residuals file; this file is named *case*.*rsr*. Entries in this file are the same as those in the residuals file for observation groups to which a covariance matrix is not assigned. However for those to which a covariance matrix is assigned "rotated residuals" and "rotated weights" (and functions thereof) are represented instead of the real residuals and weights.

Let $\mathbf{C}(\boldsymbol{\varepsilon})$ be a user supplied observation covariance matrix. The objective function component for the observation group to which the covariance matrix is assigned is calculated using equation 3.7.2 which is now repeated as equation 5.3.4.

$$\Phi = \mathbf{r}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{r} \tag{5.3.4}$$

Through singular value decomposition, $\mathbf{C}(\boldsymbol{\varepsilon})$ can be written as

$$C(\varepsilon) = \mathbf{EFE}^t \quad (5.3.5)$$

where **E** is an orthonormal matrix and **F** is a diagonal matrix with singular values arranged in decreasing order; see Doherty (2015). Equation 5.3.4 can then be written as

$$\Phi = \mathbf{s}^t\mathbf{F}^{-1}\mathbf{s} \quad (5.3.6)$$

where the vector of "rotated residuals" **s** is calculated as

$$s = \mathbf{Er} \quad (5.3.7)$$

This is equivalent to

$$\Phi = \sum (s_i w_i)^2 \quad (5.3.8)$$

where the *i*'th w_i is the square root of the *i*'th diagonal element of $\mathbf{F}^{-1}$. s_i and w_i are the rotated residuals and weights represented in the rotated residuals file. Rotated observations and their model-generated counterparts are calculated in the same way, that is through multiplication by the **E** matrix.

Because a new set of "rotated observations" is calculated for members of this group, the user-assigned names for the original observations are no longer applicable. Hence when PEST lists the names of the new observations to this file, it formulates new observation names by adding the string "_r" to the names of the original observations. However it is important to note that a rotated observation whose name is formulated by adding the string "_r" to the name of an original observation has no more of a direct relationship to that original observation than it does to any other member of the original observation group; this observation naming convention is just a convenience.

5.3.6 Interim Residuals File

At the beginning of each iteration PEST writes a file named *case.rei* where *case* is the filename base of the PEST control file. This file is a "temporary" or "interim" residuals file. It contains six columns of data. The first two columns cite observation names and group names; then follow observation values (supplied by the user) and their current model-generated counterparts. Current residuals and weights follow that in subsequent columns.

The following should be noted.

1. Model-generated observations recorded in the residuals file pertain to the best parameters calculated at the stage of the parameter estimation process at which the interim residuals file is written.
2. If an observation belongs to a group for which a covariance matrix is supplied, then a weight cannot be assigned to it. Hence, in accordance with the convention adopted for the residuals file *case.res* recorded at the end of the parameter estimation process, the string "Cov. Mat" is recorded in place of a weight.
3. As is explained in chapter 9 of this manual, weights pertaining to members of regularisation groups change throughout the inversion process. Hence these weights are different for different *case.rei* files recorded at different stages of a regularised inversion process.

5.3.7 The Matrix File

During each iteration, just after it has calculated the Jacobian matrix, PEST calculates the "post-calibration" covariance and correlation coefficient matrices, as well as the eigenvalues

and normalised eigenvectors of the covariance matrix, for the current set of parameter values. However this only happens if the following conditions are met.

- ICOV, ICOR and IEIG are set to 1 in the "control data" section of the PEST control file;
- No form or regularisation is employed (as the use regularisation renders these matrices meaningless); hence neither singular value decomposition nor LSQR is used to solve the inverse problem, and Tikhonov regularisation is not employed;
- The matrix $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ is invertible, where **J** is the Jacobian matrix and **Q** is the weight matrix.

These matrices are written to a special file named a "matrix file". This file is named *case*.*mtt* where *case* is the current case name (i.e. the filename base of the PEST control file). Each time this file is written the previous file of the same name is overwritten. Hence the matrices contained in the matrix file pertain to the most recent iteration.

If any of ICOV, ICOR or IEIG are set to zero, the corresponding matrix is not recorded in the matrix file. If they are all set to zero, then no matrices are written to this file. If ICOV is set to 1 then, as well as recording the covariance matrix to the matrix file, PEST records current parameter values and standard deviations. (The standard deviation of a parameter is the square root of its variance; parameter variances comprise the diagonal elements of the covariance matrix.) As with the elements of the covariance and associated matrices, the standard deviation of a parameter actually pertains to the log of that parameter if the parameter is log-transformed during the parameter estimation process.

The observant PEST user may notice slight differences between the matrices recorded in the final matrix file and those recorded in the run record file at the end of the PEST run. If the lowest objective function achieved during the parameter estimation process was calculated by PEST during its final iteration, then you may expect that these two sets of matrices will be very similar. Nevertheless, there are often differences between them. These differences result from the fact that the reference variance (see equation 5.3.3) used in calculation of matrices which are recorded in the matrix file is computed using the objective function calculated at the end of the previous iteration, whereas for the covariance and related matrices recorded in the run record file, the best objective function calculated during the whole parameter estimation process is used in computing the reference variance. If the best objective function was computed on the final parameter upgrade, this will differ slightly from that calculated at the beginning of the last iteration.

5.3.8 The Condition Number File

Unless PEST is using singular value decomposition or LSQR for solution of the inverse problem, it writes the condition number of the matrix which it must invert to calculate parameter upgrades (i.e. $\mathbf{J}^t\mathbf{Q}\mathbf{J} + \lambda\mathbf{I}$) to a "condition number file". The condition number is recorded for every Marquardt lambda tested during every PEST iteration.

The condition number file is named *case*.*cnd* where *case* is the filename base of the PEST control file. If the condition number is high (generally greater than about 10^4), this signals the fact that PEST may not have been able to invert the matrix properly as a result of near or total singularity arising out of excessive parameter insensitivity or correlation. At the very least, this is an indicator of nonuniqueness in estimated parameters; at most, it means that PEST may not have been successful in lowering the objective function. In either case, you should re-formulate the inverse problem. Either hold some parameters fixed or, better still, introduce

some form of regularisation (for example singular value decomposition and/or Tikhonov) to the parameter estimation problem.

5.3.9 Singular Value Decomposition File

If the EIGWRITE variable in the "singular value decomposition" section of the PEST control file is set to 1, then PEST writes a file named *case.svd* (where *case* is the filename base of the PEST control file). It extends this file on every occasion that it computes a parameter upgrade vector. On each such occasion it records singular values (arranged in decreasing order) and corresponding singular vectors of $(\mathbf{J}^t\mathbf{Q}\mathbf{J} + \lambda\mathbf{I})$. It also records the number of singular values that are actually used in computation of the parameter upgrade vector (i.e. the number of singular values remaining after truncation of small and zero singular values). Singular value decomposition is carried out on every occasion that a Marquardt lambda is tested and a corresponding parameter upgrade vector is calculated.

Where parameters are high in number, the singular value decomposition output file can become very large. Furthermore, unless RLAMBDA1 is set to zero (which is not suggested) the information contained in this file is dependent on current values of the Marquardt lambda. Many of the utilities described in part II of this manual provide the same (or better) information as that which can be obtained from this file. Hence in most instances of PEST usage it is not worth recording.

5.3.10 LSQR Output File

If the LSQRWRITE variable in the "lsqr" section of the PEST control file is set to 1, information generated by the LSQR solver is written to a file named *case.lsq* where *case* is the filename base of the PEST control file. Fresh information is generated by the LSQR solver on each occasion that it is called by PEST; hence this file can become quite lengthy.

5.3.11 Run Management Record File

If model runs are undertaken in parallel using either Parallel PEST or BEOPEST, then PEST records a run management record file named *case.rmf*. This contains a history of communications between the PEST master and each of the slaves which parameterize separate model runs and issue the system command for these runs to be carried out.

5.3.12 Pareto Output Files

Files named *case.par.N*, *case.pod* and *case.ppd* are recorded if PEST is run in "pareto" mode. The first two of these are ASCII files while the third is a binary file. File *case.par.N* contains parameter values calculated at the end of iteration *N*. File *case.ppd*, which is continuously updated as PEST runs, contains objective functions computed during a Pareto run. File *case.ppd* is a binary file which can be accessed by Pareto-specific PEST postprocessors.

5.3.13 The Jacobian Matrix File

At any stage of the inversion process implemented by PEST, the Jacobian matrix corresponding to best parameters achieved up to that stage of the process can be found in a binary file named *case.jco*. Some of the many uses to which the contents of this file can be put are discussed in section 3.6. This file is often referred to simply as a "JCO file" in parts I and II of this manual.

5.3.14 Resolution Data File

Unless the IRES variable in the "control data" section of the PEST control file is set to 0, then if PEST employs any form or regularisation (singular value decomposition, LSQR and/or Tikhonov) it writes a "resolution data file" named *case.rsd* where *case* is the filename base of the PEST control file. (Note that if the optional IRES variable is omitted from the PEST control file, then its value is assumed to be "1").

The resolution data file is a binary file whose contents cannot be read by the user. Instead it is used by the RESPROC and related utility programs described in part II of this manual.

PEST updates the resolution data file many times during the course of the inversion process such that data contained within it always pertains to the best parameters achieved at that stage of the process; it is thus overwritten whenever estimated parameters are improved.

5.3.15 Other Files

If the RSTFLE variable in the "control data" section of the PEST control file is set to 1, then PEST writes a series of binary data files containing information that it needs in order to re-commence execution if the inversion process is prematurely terminated. See appendix B for a list of these files.

If model runs are undertaken in parallel, PEST uses binary, direct access files named *pest###.dap* and *pest###.dao* for storing entries in the parameter and observation run queues.

5.3.16 PEST's Screen Output

As well as recording the progress of the parameter estimation process to its run record file, PEST also prints some of its run-time information to the screen; through this means the user is informed of the status of this process at any time.

Unless you are using Parallel PEST or BEOPEST (where the master and slaves are run in separate command line windows), the model of which PEST has control may write its own output to the screen. If this is the case, it will interfere with PEST's presentation of run record information to the screen. Perhaps this will not worry you because it allows you to check that the model is running correctly under PEST's control; in any case, you can inspect the run record file at any time by opening it in a text editor while PEST is running. However, if you find use of the same screen by both the model and PEST annoying, you may be able to redirect model screen output to a file using the ">" symbol in the model command line; this will leave the screen free to display PEST's run-time information only. Thus, for example, if a program named *modelrun.exe* is run by PEST as the model command line, or is cited in a batch file which PEST runs through the model command line, its screen output can be directed to a file named *temp.dat* by running it using the command

```
modelrun > temp.dat
```

Alternatively the command

```
modelrun > nul
```

can be employed to run *modelrun.exe*. In the latter case screen output is simply "lost", for there is no *nul* file. In the UNIX environment, the second of the above commands should be replaced by

```
modelrun > /dev/null
```

5.3.17 Run-time Errors

As was discussed above, PEST performs limited checking of its input dataset. In the event of an error or inconsistency in its input data PEST terminates execution with a run-time error message. Unlike PESTCHEK (see part II of this manual), PEST will not continue reading its input data files in order to determine whether more errors are present so that it can list them as well; rather it ceases execution as soon as it has noticed that something is wrong.

Other errors can arise later in the inversion process. For example, if the instruction set fails to locate a particular observation, PEST will cease execution immediately with the appropriate run-time error message. This may happen if the model has varied the structure of its output file in response to a certain set of parameter values in a way that you did not anticipate when you wrote the instruction set. It may also arise if the model terminated execution prematurely because of some internal model numerical glitch. Hence if a run-time error message informs you that PEST was not able to read the model output file correctly, *you should check both the screen and the model output file for a model-generated error message. If there is a compiler-generated error message on the screen informing you of a floating point or other error, and this is followed by a PEST run-time error message informing you that an observation could not be found, then the model, not PEST, was responsible for the error.* You should then carefully inspect the model output file for clues as to why the error occurred. In some cases you will find that one or a number of model parameters have transgressed their allowed domains, in which case you will have to adjust their upper and/or lower bounds accordingly in the PEST control file.

Another model-related error which can lead to PEST run-time errors of this kind will occur if the directory (i.e. folder) which contains the model executable file is not cited in either the PATH environment variable or in the "model command line" section of the PEST control file. In this case, after PEST attempts to make the first model run, you will receive the message

```
Running model .....Bad command or file name
```

prior to a PEST run-time error message informing you that a model output file cannot be opened. (Note, however, that the model path is not required if the model executable resides in the current directory.)

It is normally an easy matter to distinguish PEST errors from model errors, as PEST informs you through its screen output when it is running the model. A model-generated error, if it occurs, will always follow such a message. Furthermore, a PEST run-time error message is clearly labelled as such, as shown in figure 5.6. If you are using Parallel PEST or BEOPEST the model window will be different from the PEST window. In this case it will be much easier to distinguish an error originating from the model from an error originating from PEST.

```
**************************************************************************
Error condition prevents continued PEST execution:-

Cannot open model output file model.out.
**************************************************************************
```

Figure 5.6 A PEST run-time error message.

PEST run-time errors are written both to the screen and to the PEST run record file.

5.4 Stopping and Restarting PEST

5.4.1 Interrupting PEST Execution

At the end of every model run PEST checks for the presence of a file named *pest.stp* in the directory (i.e. folder) from which it was invoked. If this file is present, PEST reads the first item in the file. If this item is "1", PEST ceases execution immediately. If it is "2" PEST ceases execution after it prints out parameter statistics. If it is "3" PEST pauses execution. To resume PEST execution after a pause, rewrite file *pest.stp* with a "0" as the first entry.

File *pest.stp* can be written by the user using any text editor while positioned in another window to that in which PEST is running. However this file can also be written using programs PPAUSE, PUNPAUSE, PSTOP and PSTOPST supplied with PEST simply by typing the name of the appropriate program as a command while situated in the PEST working folder in another command line window. As the names suggest, PPAUSE writes a "3" to *pest.stp* in order to tell PEST to pause execution; PUNPAUSE writes a "0" to *pest.stp* to tell it to resume execution; PSTOP writes a "1" to tell PEST to cease execution, while PSTOPSTP instructs PEST to cease execution with a full parameter statistics printout by writing a "2" to file *pest.stp*. Note that if the non-parallel version of PEST is running, PEST will not respond to the presence of file *pest.stp* until the current model run is complete. In contrast, Parallel PEST and BEOPEST respond immediately; however the various incidences of the model will continue to run to completion in their own windows after PEST execution has ceased.

There is, of course, another way to stop PEST. Simply press <Ctl-C> in the command line window in which PEST is running. If a non-parallel version of PEST is running, these keys may need to be pressed a few times in succession so that execution of both the model and PEST are terminated.

5.4.2 Restarting PEST

See the discussion on the restart switches in section 5.1.5.

5.4.3 Why Stop PEST?

It is rarely necessary for PEST to continue execution until it satisfies the termination criteria set out in the "control data" section of the PEST control file, or other termination criteria that are specific to its use in "predictive analysis", "regularisation" or "pareto" modes. On most occasions on which PEST is run, you will know yourself when the inversion process is "running out of steam". Where parameter numbers are high, the cost of continuing the process is also high, as renewed filling of the Jacobian matrix requires many model runs (especially if PEST has switched to use of three or five point finite-difference derivatives). Reasons for user-termination of PEST execution may include the following.

- It may be apparent from objective functions obtained to date that a good fit between model outcomes and corresponding field measurements will not be achieved. The model may thus require modification before re-embarking on the parameter estimation process.
- The values ascribed to estimated parameters may be unrealistic. This may be construed as a sign that over-fitting is occurring. You may decide that PEST should be run again with measures put in place to prevent this. If running in "regularisation" mode a higher value for the PHIMLIM control variable may be warranted.

- The behaviour of the objective function from iteration to iteration may be erratic. This may suggest that finite-difference derivatives are being corrupted by poor model numerical performance. You may like to test this hypothesis using the JACTEST utility before commencing another PEST run with defensive measures put in place.

5.5 If PEST Won't Work

5.5.1 General

Early versions of PEST often ran into difficulties if the inverse problem which PEST was asked to solve was ill-posed. Ill-posedness makes the $\mathbf{J}^t\mathbf{QJ}$ matrix singular, and hence non-invertible. Sometimes the use of high Marquardt lambda values rendered $(\mathbf{J}^t\mathbf{QJ}+\lambda\mathbf{I})$ invertible and allowed PEST-based inversion to proceed. However this did not alter the fundamentally non-unique nature of the inverse problem. Nor did the Marquardt lambda provide a numerically effective regularisation device. Hence it was up to the user to recognize the nonunique nature of the inverse problem, and to then identify parameters whose insensitivities or high correlation warranted their removal from the parameter estimation process. These were then fixed or tied and the PEST run repeated. Alternatively, manual or automatic user-interventional functionality provided by PEST (see chapter 6) could have been invoked to assist PEST in its handling of problem ill-posedness. However these strategies provided no guarantee that the parameter set achieved through the history-matching process was of minimum error variance.

These days things are different. If the inherent complexity of natural systems is recognized (and indeed embraced, as it must be if parameter and predictive uncertainties are to be quantified), then most inverse problems turn out to be ill-posed. Nevertheless a minimum error variance solution to these problems can often be attained with guaranteed numerical stability by running PEST with the following settings.

- Set RLAMFAC to -3;
- Use singular value decomposition or LSQR to solve the inverse problem;
- Employ Tikhonov regularisation. The ADDREG1 utility is often the easiest way to introduce Tikhonov regularisation to a PEST input dataset.

If PEST does not work with these measures in place, then the reason for its failure must be identified and rectified. A few possible causes of PEST run failure, and their solutions, are now discussed.

5.5.2 Model Output File not Found

Before it runs the model, PEST deletes the model output files listed in the "model input/output" section of the PEST control file. These are the model output files which corresponding instruction files must read. If, for some reason, the model fails to run, these model output files will not be present. PEST will then issue an error message such as that depicted in figure 5.6. This error message is nearly always a sign that the model did not run, or did not run to completion.

5.5.3 Objective Function Gradient Zero

If the sensitivities of all model outputs to all parameters are zero, PEST issues the following error message.

```
Phi gradient zero in non-frozen parameter space.
```

This situation can arise if there is only one model input file to which there is a corresponding template file, and the name provided for this model input file is incorrect. PEST then writes parameter values to one file while the model reads another. (The PESTCHEK program can detect many errors, but it cannot detect this one.) This problem is, of course, easily rectified. It simply requires that the name of the wrong model input file be replaced by the name of the correct model input file.

If there is more than one model input file cited in the "model input/output" section of the PEST control file, and if only one of these is named incorrectly, then the problem of input file misnaming is more difficult to detect. It may be revealed through zero-valued composite sensitivities of parameters which are only written to this model input file (though this will be hidden if regularisation is employed, as the very purpose of regularisation is create sensitivity where it would otherwise not exist). It may also be revealed through the fact that parameters that are only represented in this model input file are not adjusted through the inversion process.

5.5.4 Erratic Objective Function Behaviour caused by Bad Derivatives

If PEST is working well, the objective function falls rapidly at first, then falls more slowly, and then falls no further. The transition from rapid downward movement to no movement at all is usually gradational.

Sometimes, however, the objective function falls rapidly during one iteration, and then does not fall at all during the next iteration. Sometimes it rises rapidly and then falls rapidly again. This is often a sign of problematical finite-difference derivatives.

Problematical finite-difference derivatives can have any of the following causes.

- Parameter values are not written with sufficient precision to model input files;
- Numbers are passed with less than full precision between different executables cited in a batch or script file which PEST runs as "the model";
- Parameter increments are too low;
- The model is experiencing numerical difficulties;
- Model solver convergence criteria are not set tightly enough.

The integrity of finite-difference derivatives can be checked using the JACTEST utility described in part II of this manual. Steps which can be taken to mitigate the deleterious effects of model-corrupted derivatives include the following.

- Derivative increments can be increased;
- The DERINLB derivatives control variable can be used where low parameter increments are caused by low parameter values;
- Model solver convergence criteria can be tightened;
- If it is practical to do so, model numerical strategies such as adaptive time-stepping which may introduce a small amount of "numerical noise" to model outputs may be disabled;
- PEST's "split slope analysis" functionality can be invoked to detect and defend against model numerical output granularity (see section 3.5.7);
- A five-point finite-difference stencil can be used (see section 3.5.2);
- PEST's "automatic user intervention" functionality can be activated, with the DOAUI control variable set to "auid" (see section 6.4).

5.5.5 Other Factors Leading to Erratic Objective Function Behaviour

Where failure to lower the objective function, or erratic behaviour of the objective function, results from extreme problem nonlinearity rather than degraded finite-difference derivatives, this can be accommodated by placing limits on the amount that any parameter is allowed to change during any one iteration. Parameter change limits can be imposed using the FACPARMAX, RELPARMAX and ABSOLUTE(*N*) variables cited in the "control data" section of the PEST control file.

Log-transformation of parameters often makes a profound impact on PEST performance. As is discussed elsewhere in this manual, relationships between model outputs and model parameters are often more linear if the latter are log-transformed. Log-transformation of all parameters often accomplishes a similar outcome to normalizing parameters by their innate variabilities. This prevents super-sensitivity of those parameters whose values are expressed by numbers which are numerically small (as is often the case for recharge in a groundwater model). If log transformation is not possible because parameters may become zero or negative, then the SCALE control variable available in the "parameter data" section of the PEST control file can be used to reduce the sensitivities of individual super-sensitive parameters. The BOUNDSCALE variable that resides in the "control data" section of the PEST control file introduces global normalization of parameters with respect to their innate variabilities.

5.5.6 Excessive Number of Model Runs

In highly parameterized contexts parameters can number in the hundreds, thousands, or even tens of thousands. In these circumstances the number of model runs required for finite-difference derivatives calculation can quickly become excessive. While parallelisation, and the use of SVD-assist, can reduce the model-run burden considerably, there are many occasions where a number of simple steps can also be very effective in reducing this burden.

Where parameters hit their bounds the length of the parameter upgrade vector is shortened to accommodate this. This can slow the progress of the inversion process considerably. If the objective function does not fall very much on an iteration in which the parameter upgrade vector is shortened, PEST may prematurely commence use of higher order derivatives calculation in accordance with the setting of the PHIREDSWH variable residing in the "control data" section of the PEST control file. The number of model runs which PEST undertakes during the next iteration is therefore doubled (or quadrupled if a five-point derivatives stencil is employed).

Premature commencement of higher-order finite-difference derivatives calculation can be forestalled through an appropriate setting of the NOPTSWITCH variable.

Consideration should also be given to widening parameter bounds. Conceptually, Tikhonov regularisation is a better method of maintaining parameter sensibility than the use of bounds. Prevention of the overfitting that may require parameters to adopt unusual values is easily accomplished through use of an appropriate target measurement objective function. See chapter 9.

5.5.7 Discontinuous Problems

Use of the equations derived in Doherty (2015) upon which gradient-based inversion is based, is predicated on the assumption that model outputs are continuously differentiable functions of parameters. If the algorithmic basis of a model is such that this assumption is

violated, then PEST will have extreme difficulty in estimating parameters for that model. (However, it may still have some success if the dependence is continuous, if not continuously differentiable.)

If the level of discontinuity of model outputs with respect to parameters is not excessively high, then some of the strategies discussed above that can be adopted to accommodate poor numerical derivatives may be successful in accommodating discontinuities born of model-algorithmic deficiencies. However if the relationship between model parameters and model outputs is too degraded, then there is little choice but to use a so-called "global methodology" to estimate model parameters. The PEST-compatible CMAES_P and SCEUA_P global optimisers (particularly the latter) may be useful in this regard. See chapter 16 of this manual.

5.5.8 Parameter Correlation and Insensitivity

As stated above, where no form of numerical regularisation is employed (which is not recommended practice), the inversion process can founder where parameter correlation and/or insensitivity renders the $\mathbf{J}^t\mathbf{QJ}$ matrix non-invertible. "Classical" advice is to identify the parameters which are causing non-invertibility of this matrix, and then remove them from the parameter estimation process. Composite sensitivities can help identify insensitive parameters; a good rule of thumb is that if the ratio of highest to lowest composite sensitivity of parameters involved in the inversion process is greater than about 100, then the most insensitive parameters should be removed from that process.

Unfortunately, composite sensitivities do not allow identification of correlated parameters. In theory, these can be identified through analysis of the post-calibration covariance matrix, and of quantities calculated from this matrix such as the parameter correlation matrix and eigenvalues/eigenvectors of the post-calibration covariance matrix. However if parameter correlation is so high as to define a null space, then the post calibration covariance matrix cannot be obtained (because $\mathbf{J}^t\mathbf{QJ}$ is singular).

Fortunately, these considerations are irrelevant when the inversion process is undertaken using the settings provided in section 5.5.1 above. Solution of a nonunique inverse problem is easily accomplished when that problem is appropriately regularised, and when singular value decomposition or LSQR is used as the solution mechanism. The solution to the inverse problem thus obtained is not correct, as this is not possible; however it is of minimum error variance.

5.6 PEST Postprocessing

5.6.1 General

Upon completion of an inversion process, the outcomes of that process must be carefully inspected. The purpose of this section is to provide a few ideas of how best to accomplish this, and to suggest a few of the processing options available through PEST utility software described in part II of this manual.

5.6.2 Calibration as Hypothesis-Testing

For reasons discussed by Doherty (2015), the outcome of an inversion process rarely leads to estimates of "correct" parameter values. Rather, it leads to a solution of the inverse problem which approaches that of minimum error variance. As such, parameters obtained from this process provide an optimal starting point for post-calibration parameter and predictive

uncertainty analysis.

In real-world modelling practice, PEST is normally run many times in the course of seeking a minimum error variance solution to the inverse problem of model calibration. Early PEST runs often constitute a form of hypothesis-testing in which the capacity of the model to replicate real-world behaviour when provided with sensible parameters is tested. If this cannot be achieved, the hypothesis that the model constitutes a reasonable simulator of system behaviour can be rejected. Alterations to the model's boundary conditions, or to the simulator itself, are therefore warranted. An advantage of machine-based history-matching over manual history-matching is that a modeller can be sure that the fit between model outputs and field measurements is as good as can be achieved given currently–employed modelling concepts. If this fit is not acceptable, or if the parameters which support such a fit are not acceptable, then the need for model revision has been unequivocally demonstrated.

The question of what constitutes a "good fit" is important. Rarely, if ever, is the fit between model outcomes and field measurements commensurate with that expected from measurement noise. In nearly all cases of environmental modelling, so-called "structural noise" born of model imperfections dominates model-to-measurement misfit. The level of this noise is difficult or impossible to judge in advance. Furthermore, this "noise" is not random and has no covariance matrix; see chapter 9 of Doherty (2015) for a full discussion of this issue. Hence procedures that have been suggested by some for quantitative analysis of the statistics of residuals rarely have a place in calibration of environmental models. However this does not eradicate the need for detailed subjective analysis of model-to-measurement misfit. Hence model outputs and field measurements should be qualitatively compared in whatever ways are available to you – normally in ways that are as suggestive as possible of alterations that may need to be made to the model. These include, but are not limited to

- plots of model outputs and corresponding field measurements against time;
- plots of residuals in space.

A sometimes-useful strategy is to interpolate model outputs to corresponding field measurement points, and to then contour them as if they were field data. Superimposition of this plot on a contour map of the actual field data – contoured using the same contouring algorithm - may then suggest the locations and nature of model inadequacies that induce differences in these contours.

Ultimately, a model user must accept that his/her model is not a perfect simulator of real-world behaviour. For reasons outlined in Doherty (2015), a less than optimal fit between model outputs and field measurements must then be tolerated. If running PEST in "regularisation" mode, a modeller can choose his/her own level of fit using the PHIMLIM regularisation control variable; this variable sets the target measurement objective function below which PEST is not allowed to venture. Sometimes it is beneficial to set this low at first in order to determine the level of fit that PEST can actually achieve. The inevitable outcome of this strategy will be the estimation of parameter values that are not appealing, this constituting proof that "over-fitting" has occurred. A second PEST run can then be undertaken with PHIMLIM set 5 to 10 percent higher than the best measurement objective function achieved during the previous PEST run; it is normally the last 5 to 10 percent of fit which causes most unwanted parameter movement. The result will often be a much better parameter set, with little degradation in model-to-measurement fit. At the same time, the modeller has demonstrated that he/she can obtain a better fit he he/she wants to, but that he/she has made a conscious decision to eschew a better fit for the sake of a better claim to minimum error variance status of the estimated parameter field.

The acceptability or otherwise of parameter values inferred through the inversion process is always a matter of subjective judgement. White et al (2014) show that if the prediction required of a model is similar in type and location to the data against which it has been calibrated, then the surrogate roles that estimated parameters may be forced to play in promulgating a good fit between outputs of a defective model and corresponding measurements of system state may degrade the predictive performance of a model to only a small extent, if at all. However where a model must make predictions which are partly null-space dependent, then reasonableness of estimated parameter values should be preserved. This may require creative formulation of an objective function, and the acceptance of a less-than-perfect fit between model outputs and field measurements.

5.6.3 Traditional Statistics

Where the inverse problem is undertaken without the aid of a mathematical regularisation device, then regularisation must be undertaken manually. As has been discussed, this requires that recalcitrant parameters be identified and removed from the inversion process. An inspection of the post-calibration covariance matrix, and particularly its eigenvalues, can help in this regard. If the ratio of highest to lowest eigenvalue of this matrix is greater than about 10^7 then the problem is too ill-posed to be solved numerically. Parameters which feature most strongly in the eigenvector corresponding to the highest eigenvalue must then be considered for removal from the inversion process. If a single parameter dominates, then it is insensitive. If multiple parameters have seriously non-zero values in this eigenvector, then they are probably correlated. At least one of them should then be fixed.

If the post-calibration covariance matrix cannot be computed because of noninvertibility of the $\mathbf{J}^t\mathbf{QJ}$ matrix, then composite parameter sensitivities should be examined and insensitive parameters removed from the inversion process.

The EIGPROC utility documented in part II of this manual can help you to find the parameters which, because of their insensitivity and/or correlation, may be causing most problems to an unregularised inversion process.

5.6.4 Statistics Appropriate to Regularised Inversion

The SSSTAT utility provides a suite of sensitivity-related information that can inform a user of the "state-of-play" of the inversion process as far as parameter estimability and information content of observations is concerned. The GENLINPRED utility provides similar information, expanding its analysis to include parameter and predictive uncertainty if prior parameter uncertainties and predictive sensitivities are available. With availability of prior parameter uncertainties, the PREDUNC7 utility calculates a post-calibration covariance matrix. This can be subjected to singular value decomposition and eigen-analysis using PEST matrix utilities.

The SUPCALC utility computes the optimal number of dimensions in the solution space and thus, by inference, the null space.

5.6.5 Information Content of the Calibration Dataset

The INFSTAT and INFSTAT1 utilities provide information on the contributions that different elements of the calibration dataset make to estimation of parameters. If predictive sensitivities are available, PREDUNC5 extends this to predictions.

The SUPOBSPAR and SUPOBSPAR1 utilities describe the flow of information from the

calibration dataset to the parameter solution space.

See part II of this manual for documentation of the above utilities, and of many other utilities that can add value to the outcomes of an inversion process undertaken by PEST.

5.6.6 Model Outputs based on Optimal Parameter Values

Before it terminates execution, PEST undertakes a final model run using optimised parameter values. Thus, at the end of a PEST run, all model input files contain optimised parameters, and all model output files contain model outputs calculated on the basis of these parameters.

Note, however, that PEST will not undertake this final model run under the following circumstances:

- if its execution is terminated using the "stop-without-statistics" utility PSTOP, or if you press <Ctl-C>;
- if it terminates execution because it encounters an error condition;
- if it is undertaking SVD-assisted parameter estimation;
- if it is being run as Parallel PEST or BEOPEST and it is terminated using the "stop-with-statistics" utility PSTOPST.

The reason for PEST's failure to conduct a final model run in the last of these cases is based on the fact that all slaves are normally busy when a user issues the directive to PEST to terminate execution. Hence a final model run cannot be immediately allocated to any slave. However if Parallel PEST terminates execution under normal conditions, the final model run is undertaken by the fastest available slave; see the run management record file if you wish to be informed which slave this is. The working directory used by that slave will then contain the model input and output files written on the basis of optimised parameter values.

A final model run using estimated parameters can also be undertaken manually if desired. There are a number of ways to do this, the easiest being as follows.

1. Use the PARREP utility to create a new PEST control file which is identical to the existing PEST control file, but with optimised parameter values replacing initial parameter values in the "parameter data" section of this file. (As discussed above, optimised parameter values are available through the parameter value file; this is named *case.par*.) If desired, regularisation can be removed from this file using the SUBREG1 utility.
2. Edit the new PEST control file, setting NOPTMAX (first variable on the eighth line of the "control data" section of this file) to 0; thus PEST will undertake only one model run, calculate the objective function, and will then cease execution. Alternatively, set NOPTMAX to -2 to calculate a Jacobian matrix, or to -1 to calculate a Jacobian matrix and then provide a full end-of-PEST-run printout.
3. Run PEST using the new control file.

6. Intervention

6.1 Introduction

With the introduction of modern regularisation methods to the PEST inversion algorithm, the need for a user to intervene in the running of PEST, or for "automatic user intervention" to accomplish similar goals, has been largely eradicated. The contents of the present chapter can therefore be ignored by most users. However because the functionality described in this chapter has been retained in PEST, it is still described. It is not impossible that difficult cases may warrant its use, and/or that automatic user intervention can be useful in mitigating the effects of corrupted derivatives on the inversion process; see section 6.4 for a discussion of this last topic.

6.2 User Intervention

6.2.1 Aberrant PEST Behaviour in the Absence of Regularisation

Where many parameters are being estimated and some are far more insensitive than others, it is commonplace to encounter problems in an unregularised parameter estimation process. (Regularisation can include use of singular value decomposition or LSQR as a solution mechanism for the inverse problem, and/or the use of Tikhonov constraints.) Under these circumstances PEST, in response to the relative insensitivity of certain parameters, may calculate an upgrade vector in which these insensitive parameters are adjusted by a large amount in comparison to other, more sensitive, parameters. PEST may feel that this large adjustment of insensitive parameters is warranted if alterations to their values are to have any effect on the objective function; meanwhile the magnitude of this calculated adjustment may exceed the range of the linearity assumption on which basis it was calculated. Meanwhile the magnitude of the change that can be incurred by any parameter during any particular iteration of the parameter estimation process is limited by the values assigned to the PEST control variables RELPARMAX, FACPARMAX and/or ABSPARMAX(*N*). PEST reduces the magnitude (but not the direction – unless the UPVECBEND control variable is set to 1) of the parameter upgrade vector such that no parameter undergoes a change that exceeds these limits. Unfortunately, if a particular insensitive parameter dominates the parameter upgrade vector, then restricting the magnitude of this vector such that the change in the value of the insensitive parameter is limited to RELPARMAX, FACPARMAX or ABSPARMAX(*N*) (depending on its PARCHGLIM setting) results in much smaller changes to other, more sensitive, parameters. Hence, the objective function may be reduced very little (if at all).

Under these circumstances, increasing RELPARMAX, FACPARMAX and/or ABSPARMAX(*N*) is not necessarily the solution to the problem, for parameter change limits are necessary to avoid unstable behaviour in the face of problem nonlinearity.

In normal PEST usage the occurrence of this problem is easily recognised by the fact that the maximum relative, factor or absolute parameter changes for a particular iteration (as printed to the screen and to the run record file) are equal to RELPARMAX, FACPARMAX or ABSPARMAX(*N*) respectively, and that the objective function is reduced very little. PEST records the names of parameters that have undergone the largest relative, factor and (where applicable) absolute changes at the end of all of its iterations. More often than not, an inspection of the parameter sensitivity file (see section 5.3.3) reveals that these same

parameters possess low sensitivities.

6.2.2 Fixing the Problem

As stated above, this problem is most easily fixed by introducing regularisation to the inverse problem.

Another (older and no longer recommended) solution to this problem is to hold parameters which are identified as being troublesome at their current values, at least for a while. With such recalcitrant parameters "out of the road", PEST can often achieve a significant improvement in the objective function. Such temporarily held parameters can then be brought back into the parameter estimation process at a later date.

It may be found that quite a few parameters need to be held in this manner, for once a particular troublesome parameter has been identified and held, it may be found that the problem does not go away because another insensitive parameter then dominates the parameter upgrade vector. When that parameter is held, yet another troublesome parameter may be identified, and so on. All such parameters can be temporarily held at their current values if desired. A user may hold such parameters one by one as they are identified in the manner described above, or he/she may prefer instead to take the pre-emptive measure of temporarily holding at their current values all parameters identified in the parameter sensitivity file as being particularly insensitive, and hence potentially troublesome.

6.2.3 The Parameter Hold File

After it calculates the Jacobian matrix, and immediately before calculating the parameter upgrade vector, PEST looks for a file named *case.hld* (where *case* is the filename base of the PEST control file) in its current directory. If it does not find it, PEST proceeds with its execution in the normal manner. However if it finds such a file, it opens it and reads its contents in order to ascertain the user's wishes for the current iteration. Note however that PEST will ignore the presence of a parameter hold file under the following circumstances:

- Singular value decomposition or LSQR are employed for solution of the inverse problem;
- The NOPTMAX control variable is set to 0 or -2.

A parameter hold file is shown in figure 6.1.

```
relparmax 10.0
facparmax 10.0
lambda 200.0
hold parameter thick1
# hold parameter thick2
hold group conduct < 15.0
hold group thicknss lowest 3
hold eigenvector 1 highest 2
```

Figure 6.1 A parameter hold file.

Entries in a parameter hold file can be in any order. Any line beginning with the "#" character is ignored, this being interpreted as a comment line. If any lines are in error they are also ignored, for PEST does not pause in its execution, or clutter up either its screen display or its run record file with error messages pertaining to the parameter hold file. However it does report any alterations that it makes to its behaviour on the basis of directives obtained from the parameter hold file to its run record file.

A user is permitted to alter the values of certain PEST control variables using the parameter hold file. These are RELPARMAX, FACPARMAX, LAMBDA and UPVECBEND. The INITSCHFAC, MULSCHFAC, NSEARCH, RELPREDSTP and ABSPRESTP variables which control PEST's operation when run in "predictive analysis" mode can also be altered. The syntax is shown in figure 6.1; that is, the name of the variable must be followed by its new value. *It is important to note that a parameter hold file should be deleted as soon as it is no longer required. If it is left "lying around", any lines which tell PEST to alter the value of the Marquardt lambda will prevent PEST from making its normal adjustment to lambda from iteration to iteration. This may severely hamper the progress of the parameter estimation process.*

Note that once RELPARMAX and FACPARMAX have been altered using a parameter hold file, they stay altered, even if the file is removed or the lines pertaining to RELPARMAX and FACPARMAX are subsequently deleted or commented out.

To hold a parameter at its current value while the parameter upgrade vector is being calculated, use a line such as the fourth appearing in figure 6.1, i.e. the string "hold parameter" followed by the parameter's name. (If the parameter name is incorrect, PEST simply ignores the line.) If the pertinent line is removed from the parameter hold file, or the parameter hold file itself is removed, the parameter is then free to move in later iterations of the parameter estimation process.

The format for the sixth line in figure 6.1 is

```
hold group pargpnme < x
```

where *pargpnme* is the name of a parameter group and *x* is a positive number. A line such as this directs PEST to temporarily fix any parameter in the named parameter group if the composite sensitivity of that parameter is less than the supplied number (i.e. *x*). Held parameters can be freed again later by reducing *x* (to zero if desired), by deleting this line from the parameter hold file, or by deleting the parameter hold file itself.

As is illustrated in the seventh line of figure 6.1, the *n* most insensitive parameters in a particular parameter group can be held at their current values using the command

```
hold group pargpnme lowest n
```

where *n* is a positive integer. Such held parameters can be freed later in the parameter estimation process by reducing *n* (to zero if desired), by deleting this line from the parameter hold file, or by deleting the parameter hold file itself.

The syntax of the eighth line of figure 6.1 is

```
hold eigenvector n highest m
```

This option accommodates the fact that an observation dataset can be insensitive to certain groups of parameters varied in a specific ratio, even though observations might be individually sensitive to each parameter comprising the group. This is a manifestation of the phenomenon of parameter correlation. The damage done to the parameter estimation process by such an insensitive parameter combination can be every bit as bad as that done by insensitive parameters on their own. Hence, in some circumstances, the parameter estimation process may benefit from the temporary removal of some or all members of such a damaging parameter combination.

In interpreting the above parameter hold command, eigenvectors of the current parameter covariance matrix are counted in order of the magnitude of their corresponding eigenvalues,

starting from the highest eigenvalue and working down. Thus eigenvector number 1 is the eigenvector associated with the highest eigenvalue of the covariance matrix. (This will be the eigenvector most commonly cited in parameter hold files supplied by the user, because the magnitude of an eigenvalue is a measure of the insensitivity of the objective function to parameters varied in ratios specified by the elements of the corresponding eigenvector.)

Once the desired eigenvector has been identified (i.e. eigenvector number *n* in the above command), PEST then selects the parameters which comprise the *m* largest components of that eigenvector. These are the parameters which are, collectively, those to which the observation dataset is least sensitive. PEST then holds these parameters at their current values while it calculates the parameter upgrade vector.

As is discussed in section 5.3.7, eigenvalues and eigenvectors are available at all stages of the parameter estimation process through the matrix file recorded by PEST during every iteration (if no form or regularisation is included in the inversion process).

While the ability to hold parameters according to their component magnitudes in various eigenvectors can be of use in difficult cases, care should be taken in using this functionality. Where excessive parameter correlation is damaging the parameter estimation process, it is often sufficient for only 1, 2 or a very few of the correlated parameters to be held, rather than all of them, when calculating an upgraded parameter set. The parameters to which the held parameters were formerly correlated are then free to move as PEST searches for the objective function minimum using the reduced parameter set. This may result in a greater objective function improvement than if all of the correlated parameters are held. Unless all of the held parameters have values close to optimal (with "optimal" in this case covering a broad range of values by virtue of their correlated state) there will be little benefit in holding *all* of them, for at least some of them will need to be assigned different values if the objective function is to be reduced.

6.2.4 Re-calculating the Parameter Upgrade Vector

In normal PEST operation, the user will probably not be aware that his/her intervention is required until after at least one iteration has elapsed. Even if it has met with little success in lowering the objective function, PEST moves on to the next iteration commencing, once again, its time-consuming calculation of the Jacobian matrix (possibly having switched to the use of high-order derivatives derivatives).

The functionality exists within PEST to halt its execution at any time and then restart its execution at that place at which it last commenced calculation of the parameter upgrade vector, i.e. at the place at which it last completed its calculation of the Jacobian matrix. Provided the PEST control variable RSTFLE is set to 1, PEST stores the current Jacobian matrix (along with other information) in a binary file each time it is calculated; the most recent Jacobian matrix is easily retrieved if PEST is asked to re-calculate the parameter upgrade vector. (Note that the most recent Jacobian matrix may not be that which is stored in the JCO file; the latter file contains the Jacobian matrix associated with the best parameters achieved so far during the inversion process.)

Re-commencement of PEST execution for upgrade vector re-calculation is effected by running PEST using the command

```
pest case /j
```

where *case* is the current PEST case name, i.e. the filename base of the PEST control file; "j" stands for "Jacobian". The same protocol is employed if restarting Parallel PEST or

BEOPEST. Whether PEST was terminated while testing the efficacy of different Marquardt lambdas in lowering the objective function, or whether it was terminated after the iteration counter had "ticked over" and PEST was engaged in calculation of a new Jacobian matrix, PEST will re-commence execution at the place at which the last Jacobian matrix had just finished being calculated. Thus, depending on what PEST was doing when its execution was terminated, it may re-commence execution within the same iteration as that in which it was interrupted, or in the previous iteration. In either case, it moves straight into calculation of the parameter upgrade vector and the testing of different Marquardt lambdas.

It is through this restart mechanism that user-assistance becomes possible with PEST. Upon inspection of the run record file and the parameter sensitivity file, a user may decide that PEST can do better in improving the objective function if it attempts the last parameter upgrade again with certain parameters, or groups of parameters, held fixed. Thus the laborious calculation of the Jacobian matrix is not wasted, for PEST is able to get a "second chance" at using this important information in calculating a better parameter set.

PEST can be stopped and restarted using the "/j" switch as many times as is desired. Thus you can progressively hold more and more parameters fixed until a significant improvement in the objective function is realised. Then you can let PEST move on to the next iteration.

6.2.5 Maximum Parameter Change

As has already been discussed, at the end of each optimisation iteration, PEST records on its run record file the maximum factor and relative changes undergone by any parameter (as well as pertinent absolute changes if ABSOLUTE(*N*) parameter limiting is active). It also records the names of the parameters undergoing these maximum changes. This, in combination with the contents of the parameter sensitivity file, may help you to decide which (if any) parameters to temporarily hold at their present values using directives supplied in the parameter hold file.

6.3 Automatic User Intervention

6.3.1 Concepts

The previous section describes PEST's user-intervention functionality. As is discussed in that section, when the unregularised parameter estimation process undertaken by PEST appears to be going nowhere, the situation can often be remedied by selectively withdrawing certain parameters (normally the most insensitive ones) from the parameter estimation process for a while. With these parameters temporarily held at their current values (and thus not requiring adjustment), the so-called "normal matrix" (i.e. the $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix) which PEST must invert in order to determine the parameter upgrade vector is often much better conditioned. Furthermore, because the necessity to impose relative, factor or absolute limits on the often broad movements of these insensitive parameters is now lifted, the unencumbered fruitful movement of more sensitive parameters can take place.

This process has been automated in PEST; as a consequence, PEST's unregularised performance in difficult parameter estimation contexts can be improved. (However, as was discussed above, the introduction of regularisation to the inverse problem would almost certainly promote greater improvements.)

"Automatic user intervention" is only available if PEST is working in "parameter estimation" mode. If PEST is being used in "predictive analysis", "regularisation" or "pareto" modes,

then intervention must be manual.

It is also important to note that the automatic user intervention process commences afresh in each new iteration. That is, all parameters which were temporarily held at their current values during any one iteration are free to move during the following iteration. In many cases the same parameters will then be held again; however the decision to hold any parameter during any particular iteration is based solely on its sensitivity as calculated during that iteration, and not on the sensitivity history of the parameter during previous iterations.

Before describing PEST's automatic user intervention functionality in detail, it should be pointed out that truncated singular value decomposition as a solution process for an ill-posed inverse problem plays a similar role to that of automatic user intervention; however it is far more effective and far more efficient than automatic user intervention in stabilizing the solution process of that problem. Like automatic user intervention, singular value decomposition has the capacity to identify and remove recalcitrant parameters from the parameter estimation process. However singular value decomposition is stronger than automatic user intervention in that it can also remove recalcitrant parameter *combinations* from this process while maintaining freedom of movement of combinations of parameters that are orthogonal to those which are temporarily held in place. This overcomes the deleterious effects on the inversion process of high levels of parameter correlation and/or the existence of a null space. See Doherty (2015) for details.

6.3.2 Automatic User Intervention Control Variables

An optional section (named "automatic user intervention") can be included in a PEST control file. If this section is not present in a particular PEST control file PEST can still undertake automatic user intervention; however it will use default values for all of the variables which govern this process. Like all other sections of the PEST control file, the "automatic user intervention" section must be led by a section header, the words comprising the header being preceded by a "*" character followed by a space. If present, the "automatic user intervention" section of the PEST control file must follow the "control data" section and precede the "parameter groups" section. An example is provided in figure 6.2.

```
pcf
* control data
restart estimation
19  19    2     0    3
2    3    single point  1  0  0
5    2    0.03  0.03 10
3.0  3.0  0.001  2    0
0.1  aui
30   0.01 3     3    0.01  3
1    1    1
* automatic user intervention
8    1    .9   0
10   0     3
0.8  0.95  3
* parameter groups
ro relative 0.001 0.0001 switch 2 parabolic
hhh relative 0.001 0.0001 switch 2 parabolic
* parameter data
ro1  none relative 1.000000 0.1 10000 ro 1 0
ro2  none relative 1.000000 0.1 10000 ro 1 0
ro3  none relative 1.000000 0.1 10000 ro 1 0
etc
```

Figure 6.2 Part of a PEST control file which contains an "automatic user intervention" section.

The names of the variables featured in the "automatic user intervention" section of the PEST control file, and their locations within this section, are depicted in figure 6.3.

```
* automatic user intervention
MAXAUI       AUISTARTOPT     NOAUIPHIRAT   AUIRESTITN
AUISENSRAT   AUIHOLDMAXCHG   AUINUMFREE
AUIPHIRATSUF AUIPHIRATACCEPT NAUINOACCEPT
```

Figure 6.3 Specifications of the "automatic user intervention" section of a PEST control file.

Automatic user intervention can be activated or deactivated using a special variable which resides in the "control data" section of the PEST control file. This resides on the seventh line of the "control data" section of the PEST control file, following the PHIREDSWH variable. Its name is DOAUI. This is a text variable that must be supplied as either "aui" or "noaui" (see figure 6.2). If it is supplied as "aui", PEST will undertake automatic user intervention; if there is no "automatic user intervention" section present in the PEST control file, default values are used for all automatic user intervention variables. If it is supplied as "noaui", the "automatic user intervention" section of the PEST control file, if present, are ignored. If no value is supplied for this variable, then a value of "noaui" is assumed. However if the same PEST control file contains an "automatic user intervention" section, PEST will cease execution with an appropriate error message.

The role of each of the variables appearing in the "automatic user intervention" section of a PEST control file is now described in detail.

MAXAUI

During each iteration PEST first calculates the Jacobian matrix. Then it calculates a parameter upgrade vector in which no automatic user intervention is attempted. It is at this point that automatic user intervention can be introduced to attempt calculation of an improved parameter upgrade vector. On the first such attempt, a certain parameter is held at

its current value and the parameter upgrade vector recalculated; on subsequent attempts, more parameters are held at their current values. Each such cycle in which an additional parameter is held and the parameter upgrade vector is recalculated is referred to as an "automatic user intervention iteration" (or "AUI iteration"). MAXAUI designates the maximum number of AUI iterations that can be carried out during any one iteration of the parameter estimation process.

MAXAUI is an integer variable. If you do not want PEST to undertake automatic user intervention at all, set it to zero. Its default value is 0.75 times the number of parameters which are neither tied nor fixed.

AUISTARTOPT

This is the PEST iteration in which the automatic user intervention process commences. Sometimes it is desirable for PEST to spend some time in attempting to adjust all parameters (no matter how insensitive they appear to be) without any of them being held through the automatic user intervention process. Automatic user intervention can then be introduced after a few optimisation iterations have elapsed. In some cases, delaying the onset of automatic user intervention in this manner can reduce the possibility of certain parameters being inadvertently held at inappropriate values.

PEST's default setting for AUISTARTOPT is 1 (thus ensuring that automatic user intervention is potentially introduced on the first iteration of the parameter estimation process). However a setting of 2 or 3 (or even higher) is often appropriate. AUISTARTOPT is an integer variable.

NOAUIPHIRAT

If PEST can lower the objective function sufficiently well without holding any parameters at their current values, automatic user intervention is unnecessary. The NOAUIPHIRAT variable is used to set the threshold at which automatic user intervention is attempted. If, on any particular iteration, PEST is able to lower the objective function such that it is below NOAUIPHIRAT of its value for the previous iteration, PEST will not attempt automatic user intervention during that iteration. PEST's default value for NOAUIPHIRAT is 0.9.

There is an important interplay between NOAUIPHIRAT and PHIREDSWH. Recall from section 4 of this manual that if the objective function is not reduced by a relative amount of PHIREDSWH during any particular iteration of the parameter estimation process, then PEST will commence the use of higher order derivatives calculation during the next iteration for any parameter groups for which FORCEN is set to "switch" or "switch_5". The cost of undertaking higher order derivatives calculation instead of two-point derivatives calculation is that PEST must undertake at least twice as many model runs per iteration. Hence the onset of higher order derivatives calculation should be deferred for as long as possible. Thus NOAUIPHIRAT should be set the same as, or slightly lower than, (1.0-PHIREDSWH). An ideal setting for PHIREDSWH is 0.1. Thus NOAUIPHIRAT should be set at 0.9 or slightly lower.

AUIRESTITN

Under some circumstances it may be advisable to give the automatic user intervention process a "rest" every now and then. This allows parameters which are repeatedly held on the basis of their insensitivity to undergo some adjustment (even though the effect of this adjustment on the objective function may be only slight). AUIRESTITN (an integer) is the

automatic user intervention variable which governs the implementation of this "resting" process. Once the automatic user intervention process commences (it commences on the first occasion on or after AUISTARTOPT iterations have elapsed on which the objective function without automatic user intervention does not fall to NOAUIPHIRAT of its previous value), automatic user intervention is "rested" every AUIRESTITN iterations. Thus, for example, if AUIRESTITN is set to 3 and automatic user intervention is introduced to the inversion process for the first time during iteration number 5, then automatic user intervention will not be undertaken on optimisation iterations 7, 10, 13 etc.

PEST's default setting for AUIRESTITN is zero; that is, no automatic user intervention resting is undertaken. Note that an AUIRESTITN value of 1 is illegal as this makes no sense. AUIRESTITN is an integer.

AUISENSRAT

Sensitive parameters should not be held, for it is important that such parameters be adjusted through the inversion process. PEST will not hold any parameter as part of the automatic user intervention process if the ratio of the highest sensitivity of any adjustable parameter to the sensitivity of that parameter is lower than AUISENSRAT. In the present context, an "adjustable parameter" is a parameter which is neither tied nor fixed, nor is at its upper/lower bound, nor is already being held through manual or automatic user intervention.

The default setting for AUISENSRAT (a real variable) is 5.0.

AUIHOLDMAXCHG

This is an integer variable which must be set to either 1 or 0. As is explained above, sometimes an insensitive parameter can hamper the progress of the parameter estimation process through requiring enforcement of the FACPARMAX or RELPARMAX parameter change limit. Enforcement of either of these limits can greatly reduce the movement of other, more sensitive, parameters. The result can be a disappointingly small improvement in the objective function. If AUIHOLDMAXCHG is set to 1, PEST can "target" parameters which are incurring the enforcement of these limits when it is deciding which parameter to hold next in the automatic user intervention process (provided they are insensitive enough as determined by the value of AUISENSRAT). Alternatively, if AUIHOLDMAXCHG is set to 0, PEST will only take parameter sensitivities into account (and not the amount by which PEST has tried to adjust these parameters) when deciding on the order in which parameters should be held.

The default setting for AUIHOLDMAXCHG is 0.

AUINUMFREE

During any particular iteration, PEST will carry out no further automatic user intervention iterations (and will thus hold no further parameters at their current values) if the number of adjustable parameters has been reduced to AUINUMFREE. As is mentioned above, in the present context an "adjustable parameter" is one which is neither tied, nor fixed, nor held at its bound, nor currently held through manual or automatic user intervention.

AUINUMFREE is an integer; its default value is 3.

AUIPHIRATSUF

As was mentioned above, after it has calculated the Jacobian matrix, PEST initially calculates a parameter upgrade vector without any parameters held through automatic user intervention

(however they can be held through manual user intervention if desired). On the basis of the results of this initial parameter upgrade attempt, PEST then decides whether or not to undertake automatic user intervention.

For every AUI iteration which it undertakes, PEST calculates the ratio of the objective function achieved during that AUI iteration, to the objective function achieved without any automatic user intervention. If this ratio falls below AUIPHIRATSUF, PEST undertakes no further AUI iterations; instead it commences the next iteration of the parameter estimation process.

A suitable setting for AUIPHIRATSUF (a real variable) is 0.8. This is the default value used by PEST.

AUIPHIRATACCEPT

The temporary removal of insensitive parameters from the inversion process has the significant advantage that it often allows the process to continue unhindered when it would otherwise stall. However a possible disadvantage of temporary parameter withdrawal is that parameters which may need to be adjusted do not get sufficient opportunities to have their values altered by PEST. To alleviate this possibility, PEST will not accept the upgraded parameter set calculated during any AUI iteration unless the objective function improvement forthcoming from that AUI iteration is significant enough to make this acceptance worthwhile.

As discussed above, during every iteration of the inversion process PEST first calculates an upgraded parameter set with no parameters held at their current values (unless they are held manually by the user). The objective function so obtained is then the "current objective function". If PEST decides to undertake one or a number of AUI iterations, it will not accept the parameter values calculated during those iterations unless the objective function falls below the fraction AUIPHIRATACCEPT of the current objective function. If it does fall below this value, however, the new objective function then becomes the "current objective function". If further AUI iterations are undertaken then, once again, the objective function must fall below AUIPHIRATACCEPT of the current objective function before any new set of parameter values is accepted.

AUIPHIRATACCEPT is a real number. PEST's default value for it is 0.99.

NAUINOACCEPT

If NAUINOACCEPT AUI iterations have elapsed since the objective function has fallen below AUIPHIATACCEPT of the "current objective function" (defined in the previous paragraph), PEST concludes that the carrying out of further AUI iterations would be a fruitless activity; thus it moves on to the next iteration of the parameter estimation process. NAUINOACCEPT is an integer, the default value of which is 0.75 times MAXAUI.

6.3.3 A Note on Default Automatic User Intervention Settings

The default settings for variables which control operation of PEST's automatic user intervention functionality are such as to cause PEST to make every effort possible to lower the objective function on every iteration of the parameter estimation process. Hence the model run burden may be high and computation times may be long. The use of non-default values for some or all of these control variables may reduce the model run burden in certain circumstances. However unless model run times are long the default values are satisfactory

and are guaranteed to work well. Hence introduction of an "automatic user intervention" setting to a PEST control file is rarely required.

6.4 Bad Derivatives Mitigation

6.4.1 General

As was discussed in section 3.5 of this manual, calculation of finite-difference derivatives of model outputs with respect to adjustable parameters can be corrupted if the numerical behaviour of a model is poor. Corrupted elements of the Jacobian matrix are often recognised by the fact that their values are unusually high. However it is very difficult to know how high these values must be to qualify as being corrupted.

Another feature of corrupted derivatives is that if variation of one particular parameter causes a certain aspect of model functionality to cross an internal threshold that causes a discontinuity to arise in the relationship between that parameter and model outputs, then an entire column of the Jacobian will be corrupted because all model outputs will be affected. It follows that the parameter upgrade process may benefit from removal of this column from the Jacobian matrix. With its removal, other parameters may then be free to alter their values in ways that benefit the parameter upgrade process.

The following algorithm accommodates the presence of a corrupted column of the Jacobian matrix, mitigating its effect on parameter upgrade calculations.

1. Attempt a parameter upgrade.
2. If the objective function does not improve, identify the parameter with maximum composite sensitivity and remove that parameter from parameter upgrade calculations.
3. Attempt another parameter upgrade.
4. Repeat steps 2 and 3 until either the objective function is lowered, or until it becomes obvious that it will not be lowered through a continuation of this process.
5. During the next iteration of the parameter estimation process (when a fresh Jacobian matrix is available) repeat this process.

This sequence of steps is very similar to that discussed in the previous section through which PEST's automatic user intervention functionality is implemented. The major difference is that instead of temporarily removing parameters from the parameter upgrade process in order of increasing composite sensitivity (starting from that of lowest composite sensitivity), parameters are removed in order of decreasing composite sensitivity, starting from that of highest composite sensitivity.

6.4.2 Implementation

The above algorithm is most easily implemented simply by setting the DOAUI variable in the "control data" section of the PEST control file to "auid". However this is not allowed if PEST employs singular value decomposition or LSQR for solution of the inverse problem because PEST functionality is such that automatic user intervention is disallowed under these circumstances. Note, however, that automatic user intervention can be used successfully in this role if PEST is undertaking SVD-assisted parameter estimation and solution of the inverse problem through which values are estimated for super parameters does not employ singular value decomposition or LSQR.

If no "automatic user intervention" section is present in the PEST control file, PEST provides default values for variables which govern the operation of automatic user intervention when used for mitigation of problems caused by bad derivatives. These values are listed in table 6.1.

Variable name	Default Value with DOAUI set to "audi"
MAXAUI	3*NESPAR/4
AUISTARTOPT	1
NOAUIPHIRAT	0.9
AUIRESTITN	0
AUISENSRAT	1
AUIHOLDMAXCHG	0
AUINUMFREE	1
AUIPHIRATSUF	0.8
AUIPHIRATACCEPT	0.99
NAUINOACCEPT	3*MAXAUI/4

Table 6.1 Default values used for automatic user intervention control variables when the DOAUI variable is set to "auid". NESPAR in the above table is the number of adjustable parameters.

The AUIPHIRATSUF variable takes on a slightly different role when DOAUI is set to "auid" from that which it assumes when DOAUI is set to "aui". In the latter case it refers to the ratio of the current objective function to that achieved in the current iteration with no parameters temporarily frozen; in the former case it is the ratio of the current objective function to that prevailing at the commencement of the current iteration.

If you feel that PEST is trying to do too much automatic user intervention during any particular iteration of the parameter estimation process, the easiest way to reduce the time spent on progressively freezing parameters and re-attempting upgrade calculations is to place an "automatic user intervention" section in the PEST control file, and then assign the above default values to all variables contained therein except for MAXAUI which can be set to an appropriately low number. PEST will then limit its progressive removal of Jacobian matrix columns to MAXAUI before abandoning automatic user intervention for that iteration and moving on to its next iteration.

7. Sensitivity Reuse

7.1 Introduction

PEST's sensitivity reuse functionality provides an opportunity for reducing the model-run burden of the parameter estimation process, especially where model run times are long. However experience suggests that it should be used with caution, especially where problems are severely nonlinear. Keep in mind also, that PEST's SVD-assist functionality achieves a similar outcome, but in a more sophisticated fashion. Hence activation of PEST's sensitivity reuse functionality is not, in general, advisory. Nevertheless it is described in this manual because it is available if you want it. Most readers can skip this chapter however.

If, during one particular iteration of the (regularised) parameter estimation or predictive analysis process, PEST detects that the composite sensitivities of certain parameters are low, then it can be instructed not to waste a model run (or perhaps more model runs if higher order finite-difference derivatives computation is employed) in computing sensitivities with respect to that parameter on the next occasion that the Jacobian matrix must be filled. This can occur if PEST's sensitivity reuse functionality is activated. However, regardless of the sensitivity distribution of parameters, sensitivities for all parameters will be re-computed every N PEST iterations, where N is set by the user. (Note that Broyden updating, if activated, will still induce parameter sensitivity improvements for all parameters during every iteration, regardless of the sensitivity reuse setting.)

7.2 Implementation Details

7.2.1 Activating Sensitivity Reuse

Sensitivity reuse is activated by setting the DOSENREUSE variable on the seventh line of "control data" section of the PEST control file to "senreuse". If this is set to "nosenreuse", or simply omitted, then sensitivity reuse will not be activated.

7.2.2 Sensitivity Reuse Control Variables

A number of variables govern implementation of PEST's sensitivity reuse functionality. These can be supplied in a special "sensitivity reuse" section. Like the "automatic user intervention" section of the PEST control file, this section can be omitted if desired. In that case PEST employs default values for sensitivity reuse variables.

If present, the "sensitivity reuse" section must be placed just before the "parameter groups" section of the PEST control file. If a "singular value decomposition" section is present, the location of this section with respect to the "sensitivity reuse" section does not matter; however each of these must precede an "automatic user intervention" section if the latter is present in a PEST control file. They must also precede an "SVD assist" section if this is present.

Regardless of whether a "sensitivity reuse" section is present in the PEST control file, sensitivity reuse functionality will not be activated unless DOSENREUSE in the "control data" section of the PEST control file is present, and set to "senreuse".

The arrangement of variables within the "sensitivity reuse" section of a PEST control file is shown in figure 7.1. Following that is an example of such a section showing default values

for the variables governing its operation.

```
* sensitivity reuse
SENRELTHRESH  SENMAXREUSE
SENALLCALCINT  SENPREDWEIGHT  SENPIEXCLUDE
```

Figure 7.1 Specifications of the "sensitivity reuse" section of a PEST control file.

```
* sensitivity reuse
0.15  -1
3  -1.0  yes
```

Figure 7.2 An example of a "sensitivity reuse" section, showing default values for sensitivity reuse control variables.

The role of each of these variables is now discussed in detail.

SENRELTHRESH

SENRELTHRESH denotes the threshold of an individual parameter composite sensitivity (relative to the maximum composite sensitivity of all parameters) at which sensitivity reuse is activated for a particular parameter. That is, if the composite sensitivity of any parameter, relative to that of highest composite sensitivity, is equal to or less than this threshold, then that parameter is a candidate for sensitivity reuse on some PEST iterations.

SENRELTHRESH can be set to any non-negative value. However there is nothing to be gained by setting its value above 1.0; if it is set to 1.0 or above, then the number of parameters for which sensitivity reuse is active on any iteration is governed entirely by the SENMAXREUSE variable, as the relative composite sensitivity of all parameters is, by definition, less than the threshold.

SENMAXREUSE

This is the maximum number of parameters for which sensitivity reuse is activated on any one PEST iteration. If it is set to a negative number, then PEST automatically sets it to half the number of adjustable parameters. It should not be set to a value greater than the number of adjustable parameters minus 1. If it is set to zero, PEST will terminate execution with an error message.

If more parameters than SENMAXREUSE incur candidature for sensitivity reuse through possession of composite sensitivities less than or equal to the RELSENREUSE threshold, then parameters are eliminated from candidature in order of increasing composite sensitivity such that the total number of parameters for which sensitivities are reused is equal to SENMAXREUSE.

SENALLCALCINT

SENALLCALCINT denotes the optimisation interval at which all parameter sensitivities are recalculated, regardless of their relative composite sensitivities. If, for example, this is set to 3, then all parameter sensitivities are computed on iterations 1, 4, 7, 10 etc.

SENPREDWEIGHT

As is explained elsewhere in this manual, the composite sensitivity of a parameter is obtained by summing elements of the column of the Jacobian matrix pertaining to that parameter, with each element in a column multiplied by the square of the observation weight pertaining to

that column. If PEST is being employed in predictive analysis mode, the question then arises as to what weight the predictive sensitivity element in each column should be given. Presumably, the weight should be high enough for it to be visible in the composite sensitivity of any parameter, for getting sensitivities of the prediction right may be critical to raising or lowering that prediction as required by the predictive analysis process. Hence parameters that are critical to that prediction may not be good candidates for sensitivity reuse, for their sensitivities should be re-calculated during every optimisation iteration instead.

SENPREDWEIGHT is the weight to assign to the prediction in computation of composite parameter sensitivities when PEST is run in "predictive analysis" mode. If it is set to a negative number, PEST will compute its weight as equal to the maximum weight assigned to any other observation (including prior information if SENPIEXCLUDE is set to "no").

SENPIEXCLUDE

If a PEST control file cites prior information, then a SENPIEXCLUDE setting of "yes" will instruct PEST not to take any account of this prior information when determining composite parameter sensitivities for the purpose of deciding parameter candidates for inclusion in the sensitivity reuse process.

It should be noted. that, irrespective of any sensitivity reuse settings, regularisation observations and/or prior information equations are never included in the calculation of composite parameter sensitivities for the purpose of deciding candidature or otherwise for parameter sensitivity reuse.

8. Predictive Analysis

8.1 Introduction

When run in "predictive analysis" mode, PEST solves the constrained predictive maximisation/minimisation problem described in section 8.4 of Doherty (2015). In solving this problem PEST evaluates the maximum or minimum value that a model prediction can take while other model outputs are such that the calibration objective function is maintained at or below a user-specified value. In doing so it evaluates the post-calibration uncertainty range of the prediction. Optionally, the prediction can be accompanied by "predictive noise".

Of course, there is no reason why the prediction cannot actually be a model parameter; the post-calibration uncertainty of any parameter can thereby be explored.

The constrained maximisation/minimisation process undertaken by PEST when it is run in "predictive analysis" mode must be undertaken following a calibration process wherein an objective function is minimised. The objective function constraint imposed during the predictive maximisation/minimisation process will be somewhat greater than the minimised objective function. Theoretically, the value of the constraining objective function can be related to a predictive confidence limit using equations provided in Doherty (2015) and derived by Cooley and Vecchia (1987), Vecchia and Cooley (1987) and Christensen and Cooley (1999). In practice, however, the constraining objective function will probably be determined subjectively, as the statistics of model-to-measurement misfit will rarely be known, given the fact that in most modelling contexts it is model imperfections, rather than measurement noise, which dictates the level of fit that can be achieved between field measurements and their model-generated counterparts.

In practical terms, predictive maximisation/minimisation as a means of exploring post-calibration predictive uncertainty works well where parameters are relatively few in number and where estimation of those parameters constitutes a well-posed inverse problem. If the inverse problem of model calibration is not well-posed, then prior information which encapsulates pre-calibration expected parameter values (or linear/nonlinear relationships between expected parameter values) must form a component of the calibration dataset to make it so; such prior information may need to be accompanied by a prior covariance matrix if full expression is to be given to expert knowledge. Given the manual regularisation required for formulation of a well-posed, or almost well-posed, inverse problem, determination of the relative weighting between expert-knowledge-based prior information on the one hand and measurements of system state on the other hand then becomes a problem.

Experience demonstrates that the numerical performance of the constrained maximisation/minimisation process can be delicate. The integrity of finite-difference derivatives must be high for the process to work well – higher than it needs to be for successful minimisation of an objective function. Model numerical performance must therefore be good. While some protection against poor numerical performance can be gained through undertaking a line search in conjunction with Marquardt-lambda-based parameter upgrade testing (see below), the line search procedure is inherently serial, and hence cannot be parallelised. This limits the run times of models with which PEST can be used when undertaking predictive uncertainty analysis in this way.

The PEST suite provides other options for undertaking post-calibration predictive uncertainty

analysis, these including linear analysis, calibration-constrained Monte Carlo analysis (of which null space Monte Carlo is an example), and its Pareto functionality. All of these can work well in highly parameterized contexts where the constrained maximisation/minimisation process which PEST undertakes when run in "predictive analysis" mode may fail. Nevertheless, this mode of operation has proven itself useful in many real-world modelling contexts, and will continue to do so in the future, provided its strengths and weaknesses are properly understood.

This chapter describes how to use PEST in "predictive analysis" mode. Concepts and theory are fully described in section 8.4 of Doherty (2015).

8.2 Predictive Analysis Mode

As was discussed earlier in this manual, PEST can run in four different modes – "estimation", "regularisation", "pareto" and "predictive analysis" modes. A PEST run in "predictive analysis" mode is designed to follow a run in "estimation" mode wherein PEST minimises an objective function.

Let Φ_{min} designate the value of the minimised objective function as calculated during its previous "estimation" run. PEST's task when run in "predictive analysis" mode is to find the point identified in figure 8.2 of Doherty (2015). This is a point lying on a contour along which the objective function is Φ_0 and at which a prediction of interest has its maximum or minimum value along that contour. This point is referred to as the "critical point" in the discussion that follows. PEST finds the parameters which correspond to this point, and the value of the prediction which corresponds to this point. Φ_0, of course, must be greater than Φ_{min}. Often it is not much greater than Φ_{min} and can thus be designated as being equal to $\Phi_{min}+\delta$ where δ is a small number. The means through which a suitable value can be assigned to Φ_0 has been mentioned above and will be discussed in greater detail below.

If PEST runs in "predictive analysis" mode, then a particular model output must be identified as "the prediction" of interest. Of course, all model outputs (and parameters) that were employed in the previous "estimation" run must also be included in the PEST dataset for the "predictive analysis" run, as the objective function must be calculable just as before. However the model may need to be expanded to include the predictive output of interest. Also, it may need to be run into the future. Perhaps two versions of the model will need to be encapsulated in a single batch or script file which PEST runs through a system call; one of these may pertain to calibration conditions while the other may be run into the future to calculate the model output which must be maximised or minimised subject to an objective function constraint of Φ_0.

The predictive model output must be read like any other model output which is of interest to PEST. An instruction must be provided in an instruction file. The model output must be classified as an "observation". However it must be assigned to an observation group named "predict". Furthermore it must be the sole member of that observation group. This is how PEST recognizes the model output that it must maximise or minimise subject to calibration constraints. *It is important to note that PEST takes no notice of the "observed value" of this observation; depending on the setting of control variables discussed below, it may or may not take notice of the weight assigned to this observation. PEST's job is to maximise or minimise the model output corresponding to this observation, while maintaining the objective function at or below* $\Phi_{min} + \delta$, i.e. at or below Φ_0.

Though ostensibly different, PEST's operation in "predictive analysis" mode has much in

common with its operation in "estimation" mode. In fact, the underlying mathematics is very similar; see Doherty (2015). Like parameter estimation, the process required to determine the critical point is an iterative one, beginning at some user-supplied initial parameter set. Initial parameters can be either inside the Φ_0 contour or outside of it; in fact they can be the same initial values that were used for the preceding parameter estimation process. If they are outside of the Φ_0 contour, PEST automatically works in "estimation" mode until it is "within reach" of Φ_0, at which stage it modifies its operations to search for the critical point. It is normally good practice, however, for "predictive analysis" mode initial values to be the same as those determined through the previous "estimation" mode PEST run, and hence correspond to Φ_{min}.

When run in "predictive analysis" mode PEST still needs to calculate a Jacobian matrix, so derivatives of model outcomes with respect to adjustable parameters are still required. Derivatives can be calculated using two, three or five points; PEST can switch from one to the other as the solution process progresses. Parameters must still be assigned to groups for the purpose of assigning variables which govern derivatives calculation. Parameters can be bounded, log-transformed, linked to one another, or fixed in "predictive analysis" mode just as in "estimation" mode. In fact parameter bounds, transformations and linkages must be the same for a predictive analysis run as they were for the preceding parameter estimation run in which Φ_{min} was determined, for the value supplied to PEST for Φ_0 must be consistent with the previously determined value of Φ_{min}.

Just as in "estimation" mode, a Marquardt lambda is used to assist PEST in coping with model nonlinearities when it is run in "predictive analysis" mode; this lambda is adjusted by PEST as the optimisation process progresses. The same user-supplied control variables affect PEST's lambda adjustment procedure as when it is used in "estimation" mode (plus a couple more – see below). However, if desired, a line search procedure along the direction of the parameter upgrade vector can be used to improve calculation of the maximum or minimum model prediction for any value of the Marquardt lambda. This, in fact, is the recommended procedure.

When run in "predictive analysis" mode PEST can be stopped and restarted at any time; it can be re-started using the "/r", "/j" or "/d" switches just as in "estimation" mode (provided that the PEST RSTFLE variable was set to "restart" on the previous run); furthermore, if run as Parallel PEST or BEOPEST it can be restarted using the "/s" switch as well. Relative, factor and absolute change limits are just as important when PEST is used in "predictive analysis" mode as when it is used in "estimation" mode. Prior information can also be used; naturally if it is used in a predictive analysis run following a parameter estimation run, the prior information equations and weights should be the same for both runs.

Observation and observation group functionality in "predictive analysis" mode is identical to that in "estimation" mode. However, as was mentioned above, when PEST is used in "predictive analysis" mode there must be at least two observation groups, one of which is named "predict". This group must contain only one observation (of any name), this being the observation corresponding to the single model output for which a maximum or minimum is sought within the constraints of Φ_0. All other observations for this PEST run must be identical in value and weight to those used in the previous parameter estimation run in which Φ_{min} was determined. The weight assigned to the single observation belonging to the "predict" group (which, obviously, does not figure in the previous PEST calibration run) may or may not matter; as is discussed below, this depends on whether "predictive noise" is taken into account in the constrained predictive maximisation/minimisation process.

When run in "predictive analysis" mode, more model runs are normally required to achieve solution convergence than are required when PEST is run in "estimation" mode because it is usually a more difficult matter to find the so-called "critical point" at which a prediction is maximised/minimised while the objective function is maintained at a value of Φ_0 than it is to minimise the objective function to find the value of Φ_{min}.

PEST screen output is slightly different when run in "predictive analysis" mode from its screen output when run in "estimation" mode in that the value calculated for the prediction is written to the screen on the occasion of every attempted parameter upgrade. The user should bear in mind when monitoring PEST performance through watching its screen output, that successful PEST execution is no longer measured in terms of how much it can reduce the objective function. It is now measured by how high or low (depending on the user's request) the prediction can be made, while keeping the objective function as close as possible to Φ_0.

After completion of a predictive analysis run, the highest or lowest model prediction for which the objective function is equal to or less than Φ_0 is recorded on the PEST run record file. Corresponding parameter values are also recorded on this file as well as in the parameter value file *case.par* where *case* is the filename base of the PEST control file.

To summarize what has been discussed above, predictive analysis should only be undertaken after PEST has been used to undertake parameter estimation with the model run under calibration conditions alone. The predictive analysis and parameter estimation runs are thus closely related. All parameters, parameter transformations, parameter linkages, observations, observation weights, prior information equations and prior information weights must be the same for the two runs in order to ensure consistency of objective function values.

8.3 Predictive Analysis Control Variables

8.3.1 The PESTMODE Variable

For PEST to run in "predictive analysis" mode the following must happen.

1. The PESTMODE variable in the "control data" section of the PEST control file must be set to "prediction".
2. A "predictive analysis" section must be placed at the end of the PEST control file. This should be placed after the "prior information" section; if there is no prior information then the "prior information" section of the PEST control file can be omitted or simply left empty.
3. At least two observation groups must be defined, one of which should be named "predict".
4. This "predict" group must contain only one member.

8.3.2 Predictive Analysis Section of the PEST Control File

Specifications of the "predictive analysis" section of the PEST control file are provided in figure 8.1; as usual, optional variables are placed in square brackets. An example of a PEST control file in which PEST is asked to run in "predictive analysis" mode is provided in figure 8.2.

```
* predictive analysis
NPREDMAXMIN [PREDNOISE]
PD0 PD1 PD2
ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH
ABSPREDSWH RELPREDSWH
NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP
```

Figure 8.1 Specifications of the "predictive analysis" section of a PEST control file.

```
pcf
* control data
restart prediction
5 10 2 0 3
3 3 single point 1 0 0
5 2 0.3 0.01 10
2 3 0.001
0.1
30 0.01 5 5 0.01 5
1 1 1
* parameter groups
ro relative 0.001 0.0001 switch 2 parabolic
hhh relative 0.001 0.0001 switch 2 parabolic
* parameter data
ro1 log factor 4.000000 1e-10 10000 ro 1 0 1
ro2 log factor 5.000000 1e-10 10000 ro 1 0 1
ro3 log factor 6.000000 1e-10 10000 ro 1 0 1
h1 log factor  5.000000 1e-10 100 hhh 1 0 1
h2 log factor  4.000000 1e-10 100 hhh 1 0 1
* observation groups
obsgp1
obsgp2
predict
* observation data
ar1 1.21038 1 obsgp1
ar2 1.51208 1 obsgp1
ar3 2.07204 1 obsgp1
extra 5.0  0.0 predict
ar4 2.94056 1 obsgp1
ar5 4.15787 1 obsgp1
ar6 5.7762 1 obsgp1
ar7 7.7894 1 obsgp1
ar8 9.99743 1 obsgp1
ar9 11.8307 1 obsgp2
* model command line
model.bat
* model input/output
ves1.tpl a_model.in1
ves2.tpl a_model.in2
extra.tpl extra.dat
ves1.ins a_model.ot1
ves2.ins a_model.ot2
extra.ins  extra1.dat
* prior information
* predictive analysis
-1
2.75  2.84   5.5
0.00 0.005 1.0 2.0 8
0.00 0.05
4    0.0    0.005   4
```

Figure 8.2 Example of a PEST control file in which PEST is asked to run in "predictive analysis" mode.

Variables appearing in the "predictive analysis" section of the PEST control file are now discussed in detail.

NPREDMAXMIN

When PEST is used in "predictive analysis" mode, its task is to maximise or minimise the single model prediction while maintaining the objective function at or below $\Phi_{min}+\delta$ (i.e. Φ_0). If NPREDMAXMIN is set to 1, PEST maximises the prediction; if NPREDMAXMIN is set to –1, PEST minimises the prediction.

PREDNOISE

If PREDNOISE is set to zero or omitted, then predictive noise is not taken into account when maximising or minimising the prediction. Maximisation/minimisation of the prediction therefore serves to calculate the confidence interval of the prediction. The weight associated with the sole member of the observation group "predict" in the PEST control file is ignored under these circumstances.

On the other hand if PREDNOISE is set to 1 then predictive noise is taken into account when maximising or minimising the prediction. The so-called "prediction interval" of the prediction is thereby calculated. The weight associated with the sole member of the observation group "predict" is construed as being equal to the inverse of the standard deviation of predictive noise in this case.

See chapter 8 of Doherty (2015) and section 8.4 below for further discussion.

PD0

PD0 is a value for the objective function which, under calibration conditions, is considered sufficient to "just calibrate" the model. It is equal to $\Phi_{min}+\delta$, i.e. Φ_0 in the above discussion. A PEST predictive analysis run should be preceded by a parameter estimation run in which Φ_{min} is determined. The user then decides on a suitable value for δ and hence Φ_0 before supplying the latter as PD0 for a PEST predictive analysis run. Naturally PD0 should be greater than Φ_{min}. However in most circumstances it should only be a little greater. If predictive noise is assumed to be zero (see below) then the value of Φ_0 can be related to a particular predictive confidence interval using equations 8.3.13 and 8.3.14 of Doherty (2015). In the presence of predictive noise equations 8.4.2 and 8.4.3 apply. However all of these equations assume normally distributed random noise with a known covariance matrix. In most modelling contexts the level of model-to-measurement misfit is set by structural noise rather than measurement noise. This entitles the modeller to choose a value for Φ_0 which simply "makes sense under the circumstances". On many occasions it makes sense to choose a value for Φ_0 which is 5 percent to 10 percent greater than Φ_{min}.

PD1

In many modelling contexts the shape of the Φ_0 contour in parameter space is complex. If the model were linear the Φ_0 contour would be elliptical. However model nonlinearities, compounded by numerical granularity of model outputs, may make its shape far more complex than this, especially if viewed "under the microscope". Maximising/minimising the value of a prediction (whose calculated value may itself be affected by model granularity) while maintaining an objective function value of exactly Φ_0 may thus become a numerically difficult task.

To ease the numerical burden of finding a parameter set that lies exactly on this contour (especially when approaching it from the outside), PEST can be informed that it can accept an objective function value that is slightly higher than PD0. Thus PD0 is the value of the objective function that PEST must "aim for" when maximising (minimising) the model prediction; PD1 is the value that it will accept. Depending on the value of Φ_0 relative to Φ_{min}, this may be as little as 0.5 percent higher than PD0. Experience has shown, however, that when undertaking a line search in the parameter upgrade direction (see below) PD1 should only be greater than PD0 by about 0.2 percent or less. Alternatively, if you are not undertaking a line search for refinement of the parameter upgrade vector, or if PEST appears to be having difficulties in finding parameter sets which can raise or lower the prediction while keeping the model calibrated, PD1 may need to be as much as 5 percent to 10 percent higher than PD0.

PD2

When run in "predictive analysis" mode, the solution procedure used by PEST is very similar to that used in "estimation" mode. During each iteration PEST first fills the Jacobian matrix; then it calculates some trial parameter upgrade vectors on the basis of a number of different values of the Marquardt lambda. The latter is automatically altered by PEST during the course of the solution process using a complex adjustment procedure. If the current value of the objective function is above PD1, PEST adjusts the Marquardt lambda in such a manner as to lower the objective function; if the objective function is below PD1, PEST's primary concern is to raise or lower the model prediction.

After PEST has tested a few different Marquardt lambdas it must make the decision as to whether to continue calculating parameter upgrade vectors based on new lambdas or whether it should move on to the next optimisation iteration. If the objective function is above PD1 this decision is made using the same criteria as in normal PEST operation; these criteria are based on the efficacy of new lambdas in lowering the objective function. An important variable in this regard is PHIREDLAM. As is explained in section 4.2.6 of this manual, if PEST fails to lower the objective function by a relative amount equal to PHIREDLAM on successive parameter upgrade attempts using successive Marquardt lambdas, PEST will move on to the next optimisation iteration.

When PEST is run in "predictive analysis" mode, as the objective function approaches PD0 the relative change in the objective function, Φ, between Marquardt lambdas may be small; however the relative reduction in $(\Phi - \Phi_0)$ (i.e. the objective function minus PD0) may be sufficient to warrant testing the efficacy of another Marquardt lambda. The objective function value at which PEST stops testing for a relative objective function reduction, and begins testing for a relative reduction in $(\Phi - \Phi_0)$ is PD2. Generally this should be set at 1.5 to 2 times PD0. In either case the decision as to whether to try another lambda or move on to the next optimisation iteration is made through comparison with PHIREDLAM.

ABSPREDLAM and RELPREDLAM

During each iteration, after it has filled the Jacobian matrix, PEST tests the ability of a number of different values of the Marquardt lambda to achieve its objective. Its exact objective depends on the current value of the objective function Φ. If the objective function is above PD1, PEST's highest priority is to lower it; if the objective function is less than PD1, PEST's highest priority is to raise or lower (depending on the value of NPREDMAXMIN) the model prediction. In either case, PEST is constantly faced with the decision of whether to

test more lambdas or to move on to the next iteration.

If the objective function is below PD1 and successive Marquardt lambdas have not succeeded in raising (lowering) the model prediction by a relative value of more than RELPREDLAM or by an absolute value of more than ABSPREDLAM, PEST will move on to the next optimisation iteration. Due to the fact that the approach to the critical point is often slow, these values may need to be set low. A value of 0.005 for RELPREDLAM is often suitable; the value for ABSPREDLAM depends on the context. If you would like one of these variables to have no effect on the predictive analysis process (which is mostly the case for ABSPREDLAM), use a value of 0.0.

INITSCHFAC, MULSCHFAC and NSEARCH

When undertaking predictive analysis, PEST calculates a parameter upgrade vector in accordance with the theory presented in Doherty (2015). However it has been found from experience that the predictive maximisation/minimisation process can sometimes be made more efficient if PEST undertakes a line search along the direction of its calculated parameter upgrade vector each time it calculates such a vector, in order to find the exact point of intersection of this vector with the $\Phi_{min}+\delta$ contour. However this search will not be undertaken unless the objective function has fallen below $\Phi_{min}+\delta$ at least once during any previous iteration.

A line search is undertaken for each trial value of the Marquardt lambda. The maximum number of model runs that PEST will devote to this line search for any value of lambda is equal to the user-supplied value of NSEARCH; set NSEARCH to 1 if you wish that no line search be undertaken. Otherwise, a good value is 15.

When undertaking the line search, the initial model run is undertaken at that point along the parameter upgrade vector which is a factor of INITSCHFAC along the line of the distance that PEST would have chosen using the theory presented in Doherty (2015) alone. It has been found from experience that if complexities of model behaviour dictate that a line search is necessary, then it is worth doing it properly. So unless there is a good reason to do otherwise, a value of 0.2 to 0.3 is appropriate here; thus the line search begins at a point on the potential parameter upgrade line which is not too far from current parameter values. This accommodates the sometimes complex nature of the line search, where PEST must monitor both the value of the prediction and of the objective function. It is far from uncommon for variation of the prediction to be monotonic along the direction of the line search while the objective function may fall and then rise in this same direction.

In undertaking the line search, PEST moves along the parameter upgrade vector, increasing or decreasing the distance along this vector by a factor of MULSCHFAC as appropriate. A value of 1.3 to 1.7 is suitable for this variable in most cases. Then, once the $\Phi_{min}+\delta$ contour has been subtended by two different model runs, PEST uses a bisection algorithm to find the intersection point with greater precision.

As stated above, when undertaking a line search, set PD1 0.2 percent (or less) higher than PD0. It is also a good idea to implement the following strategies.

- Set ABSPREDSTP and RELPREDSTP reasonably tight. As is discussed below, these are actually the termination criteria for the predictive analysis process. However one tenth of these values constitute termination criteria for the line search.
- Start the predictive analysis process from optimised parameter values (for which the

objective function is less than PD0).

Because the line search is repeated for every different value of the Marquardt lambda tested, it can consume an inordinate number of model runs if significant Marquardt lambda adjustment is warranted. Code has been inserted within PEST that aims to reduce the number of line search runs required when testing Marquardt lambdas after the first during any one optimisation iteration. Nevertheless, any savings that can be made in reducing trial Marquardt lambdas will result in increased efficiency. Thus after you have had experience with using the predictive analyser on your particular task, you may wish to consider setting the initial Marquardt lambda (RLAMBDA1) lower or higher than you normally would, if this is where PEST seems to prefer its value to be. Alternatively, set NUMLAM to 1, so that only 1 Marquardt lambda is employed per optimisation iteration; if this strategy is adopted it is probably good practice to try a very low value for RLAMBDA1, maybe in the vicinity of 0.01 to 0.001 (or even zero).

While conducting a line search is a very time-consuming activity, experience has shown that it can be worth the effort in many circumstances; it is sometimes quite surprising how high or low calibration-constrained predictions can be. The cost of finding these extreme predictions, however, can be particularly harsh when using Parallel PEST or BEOPEST to undertake the predictive analysis process, because while Jacobian runs are parallelised, line search runs are not (because a line search is an innately serial process).

ABSPREDSWH and RELPREDSWH

In the "parameter groups" section of the PEST control file, the user informs PEST whether derivatives of model outcomes with respect to the members of each parameter group are to be calculated using two points, three points, five points, or two points at first and then three or five points later. When run in "estimation" mode, PEST makes the switch between two point and higher order derivatives calculation if it fails to lower the objective function by a relative amount equal to PHIREDSWH between successive iterations. The value for PHIREDSWH is supplied in the "control data" section of the PEST control file.

When used in "predictive analysis" mode, the role of PHIREDSWH is unchanged if the current objective function is above PD1. However if it is below PD1, PEST's decision to switch from two point derivatives calculation to higher order derivatives calculation is based on improvements to the model prediction. If, between two successive optimisation iterations, the model prediction is raised (lowered) by no more than a relative amount of RELPREDSWH or by an absolute amount of ABSPREPSWH, PEST makes the switch to higher order derivatives calculation. A setting of 0.05 is often appropriate for RELPREDSWH. The setting for ABSPREDSWH is context-dependent. Supply a value of 0.0 for either of these variables if you wish that it has no effect on the optimisation process. (On most occasions ABSPREDSWH should be set to 0.0.)

NPREDNORED

All of the variables on the sixth line of the "predictive analysis" section of the PEST control file are termination criteria.

When PEST is run in "estimation" mode, the parameter estimation process is judged to be complete when the objective function can be reduced no further, or if it is apparent that a continuation of the inversion process will reduce it very little. This is still the case if PEST, when used in "predictive analysis" mode, fails to lower the objective function below PD1. However if it has been successful in lowering it to this value (which it should be if PD0 and

PD1 are chosen to be above Φ_{min} as determined from a previous parameter estimation run), then termination criteria are based on improvements to the model prediction.

If NPREDMAXMIN is set to 1 and NPREDNORED iterations have elapsed since PEST has managed to raise the model prediction, then it will terminate execution. Alternatively if NPREDMAXMIN is set to -1 and NPREDNORED iterations have elapsed since PEST has managed to lower the model prediction, then it will terminate execution. A good setting for NPREDNORED is 4.

ABSPREDSTP, RELPREDSTP and NPREDSTP

If NPREDMAXMIN is set to 1 and if the NPREDSTP highest predictions are within an absolute distance of ABSPREDSTP of each other, or are within a relative distance of RELPREDSTP of each other, PEST will terminate execution. If NPREDMAXMIN is set to -1 and the NPREDSTP lowest predictions are within an absolute distance of ABSPREDSTP of each other, or are within a relative distance of RELPREDSTP of each other, PEST will terminate execution. A good setting for RELPREDSTP is 0.005. The setting for ABSPREDSTP is context-dependent; set ABSPREDSTP to 0.0 if you wish it to have no effect (which is normally the case). A good setting for NPREDSTP is 4.

Note that the maximum allowed number of optimisation iterations is set by the value of the NOPTMAX variable provided in the "control data" section of the PEST control file. Consider setting this higher than you would if you were running PEST in "estimation" mode.

8.3.3 User Intervention

User intervention, including the role of the parameter hold file, is discussed in section 6.2 of this manual. The predictive analysis control variables INITSCHFAC, MULSCHFAC, NSEARCH, RELPREDSTP and ABSPREDSTP can be adjusted midway through a PEST run through the user intervention process. All of these affect the line search procedure that is adopted by PEST if the NSEARCH control variable is set to a value greater than 1.

It has been found from experience that in solving difficult constrained maximisation/minimisation problems, RELPREDSTP and ABSPREDSTP may need to be set quite low, and that this may be discovered during a PEST run rather than anticipated before the commencement of the run. Recall that these variables indirectly provide termination criteria for the line search procedure as well as directly providing termination criteria for the predictive analysis process itself. Their values can be particularly important where small predictive uncertainty exists in a large predictive number (in which case the absolute, rather than the relative, termination criterion should be used), especially where line search increments are chosen to be small for better accommodation of nonlinear models.

8.4 Confidence and Predictive Intervals

8.4.1 General

Authors such as Vecchia and Cooley (1987) discuss the difference between a predictive confidence interval and a prediction interval. The former describes the "wiggle room" in a model prediction that is inherited from noise in the calibration dataset (and from parameter nonuniqueness if an inverse problem is ill-posed). This is the interval explored by PEST when run in "predictive analysis" mode when the PREDNOISE variable is set to 0 or omitted from the "predictive analysis" section of the PEST control file.

A predictive confidence interval takes no account of the fact that a model cannot replicate the operation of an environmental system exactly. Associated with every prediction that a model makes is a certain degree of "predictive noise". Part of this noise (probably the greater part) is the result of model inadequacies. Part of it is attributable to the fact that if a field measurement were made corresponding to the model prediction, then a certain amount of noise would be associated with that measurement. In establishing an interval about a model prediction that would encompasses this actual measurement at a certain level of confidence, this noise must be taken into account.

Where a prediction is of the same or similar type to measurements employed in the calibration process, the variance of predictive noise (if it can be assumed to be the same as that of measurement noise) is estimated through the calibration process. Suppose that the minimised objective function obtained through unregularised inversion is Φ_{min}. Suppose also that each measurement weight employed in the inversion process is proportional to the inverse of the standard deviation of the measurement to which it pertains. If the inverse problem is well-posed, PEST calculates a reference variance as

$$\sigma_r^2 = \Phi_{min}/(n\text{-}m) \qquad (8.4.1)$$

where n is the number of observations on which calibration is based and m is the number of estimated parameters. If the weight assigned to the i'th observation is w_i, then the standard deviation of noise associated with this observation is thus estimated as

$$\sigma_i = (\sqrt{\sigma_r^2})/w_i \qquad (8.4.2)$$

Where a prediction is of an entirely different type to measurements employed in the calibration process, then the level of noise associated with that prediction (in most cases resulting from incapacity of the model to simulate all details of system behaviour which pertain to this prediction) must be guessed.

8.4.2 One Methodology for Accommodation of Predictive Noise

One way in which the presence of predictive noise can be accommodated in the predictive uncertainty analysis process is to add an extra parameter, *e* to the existing PEST parameter set prior to undertaking that process. This parameter is actually the "predictive error" associated with the prediction that is to be maximised or minimised. The "estimated value" for *e* (which would normally be supplied as its initial value) is zero. A prior information equation would then be added to the optimisation process in which *e* was equated to zero. The weight assigned to this prior information equation would be such that when its inverse was multiplied by the square root of the reference variance, the standard deviation of predictive noise is obtained. Or, looking at it another way, the inverse of the weight assigned to this prior information equation should bear the same relationship to the predictive noise standard deviation as the inverse of observation weights employed in the calibration process bear to their respective measurement error standard deviations.

A new template file would then be prepared which would allow PEST to record the current value of *e* prior to each model run. Alternatively, the value of *e* could be written to an existing model input file using a slightly modified existing template file; the model would then need to be modified to read this value. Functionality would then be added to the model (or to a model postprocessor) through which *e* was added to the model-calculated prediction. This new noise-augmented prediction would then be read by PEST as the quantity to be maximised or minimised through the predictive analysis process. In maximising/minimising the prediction within the objective function constraints allowed to it, PEST would then have

the choice of increasing/decreasing the actual model output corresponding to this prediction, or the error associated with this prediction or, more likely, a combination of the two.

8.4.3 An Alternative Methodology

If the PREDNOISE variable in the "predictive analysis" section of the PEST control file is set to 1, the constrained maximisation/minimisation algorithm implemented by PEST when run in "predictive analysis" mode is enhanced so that the same outcomes as the above process can be achieved without making significant modifications to the PEST predictive analysis input dataset, and without needing to augment the model's functionality such that it adds noise to the selected predictive output. This methodology is based on theory presented in Vecchia and Cooley (1987) and described by Doherty (2015) for achieving this aim.

With PREDNOISE set to 0, predictive maximisation/minimisation is based on equations 8.4.5 and 8.4.6 of Doherty (2015). Where PREDNOISE is set to 1, equation 8.4.7 replaces 8.4.6. Meanwhile the actual predictive error *e* is calculated using equation 8.4.8.

8.4.4 Objective Function Constraints

Vecchia and Cooley (1987), Christensen and Cooley (1999) and Doherty (2015) discuss the difference between simultaneous (or Scheffé) confidence/prediction intervals and individual confidence/prediction intervals. A *simultaneous* parameter confidence region is an *m*-dimensional region in parameter space (where *m* is the number of estimated parameters) bounded by a surface of constant objective function value (which is an ellipsoid for a linear model) within which, at a nominated level of confidence, all parameters lie. The projections of this surface onto different parameter axes define the Scheffé confidence intervals for these parameters. The Scheffé 1-α confidence interval for a prediction is defined as the interval between the maximum and minimum prediction that can be made by the model using parameters that lie within the joint parameter confidence region pertaining to a nominated 1-α probability level. The simultaneous prediction interval includes predictive noise as a notional parameter in this process, and as an augmenter of the prediction, in the manner described above.

The 1-α *individual* confidence interval of a parameter is simply the range of values, centered on the best parameter estimate, within which the parameter lies at a confidence limit of 1-α, irrespective of values taken by other parameters. The 1-α individual confidence limit of a prediction is defined as the range of predictive values for which the combination of parameters that gives rise to the prediction is within the 1-α confidence interval for that combination of parameters. Individual predictive intervals take predictive error into account as both a notional parameter and a prediction augmenter as discussed above. Individual confidence and predictive intervals are smaller than simultaneous intervals at the same level of confidence.

Values of Φ_0 for 1-α simultaneous confidence and prediction intervals are given by equations 8.3.13 and 8.4.2 respectively of Doherty (2015). Meanwhile equations 8.3.14 and 8.4.3 provide values of Φ_0 for individual (1-α) predictive confidence and prediction intervals respectively.

As stated earlier, the theory on which the above equations for Φ_0 is based is inapplicable where model-to-measurement misfit is an outcome of model imperfections (which it generally is). In practice, selection of a suitable value for Φ_0 will be a subjective matter.

8.4.5 Implementation in PEST

Exploration of the confidence interval for a particular prediction requires that PREDNOISE be set to zero and that PEST be run in "predictive analysis" mode. Exploration of a prediction interval is only slightly more complex. In this case PREDNOISE must be set to 1. The weight w_e assigned to the observation comprising the sole member of the observation group "predict" must then be correct in terms of its characterization of predictive noise in the current calibration context, for PEST does not ignore this weight when calculating a prediction interval as it does when calculating a confidence interval; instead it becomes the w_e described in section 8.4 of Doherty (2015). As discussed above, this weight must bear the same relationship to predictive noise as observation weights bear to measurement noise.

With PREDNOISE set to 1, PEST's reporting of the predictive maximisation/minimisation process is altered in the following ways.

1. The reported objective function is now the normal model-to-measurement misfit objective function plus the contribution from predictive error, i.e. $(ew_e)^2$.
2. The reported prediction is now the model output specified as the prediction, plus the predictive noise term e.
3. The current value of e is reported with the prediction as the optimisation process progresses.

In all other respects, operation of PEST is outwardly the same when PREDNOISE is set to 1 as it is when PREDNOISE is set to 0.

Limited experience to date in comparing the performance of PEST's internal accommodation of the predictive noise term with the more lengthy procedure described in section 8.4.2 suggests, paradoxically, that the latter is slightly more efficient, and provides slightly higher/lower maximised/minimised predictions. It is thought that this is due to better accommodation of nonlinear problems, particularly in implementing the line search, when the e parameter is re-calculated at each stage of this search, instead of being re-calculated only when a different Marquardt lambda is employed in the parameter upgrade process as occurs for internal implementation of this procedure. However, differences between the two outcomes do not appear to be great.

8.5 Model-Based Hypothesis Testing

The predictive analysis process discussed above works well where the number of adjustable parameters is not large and where the inverse problem is well-posed. Its numerical performance deteriorates as parameter numbers rise. Furthermore, as stated above, where a problem is ill-posed, prior information must be included in the calibration dataset to render it well posed. The balancing of prior information weights against observation weights may then be problematical.

As will be discussed in the next chapter, when PEST is run in "regularisation" mode it is able to adjust prior information weights such that they are optimally balanced against measurement weights. Doherty (2015) discusses how this can form the basis of a type of predictive analysis process that can also be characterized as model-based hypothesis-testing.

The principle is similar to that described in the present chapter in that the model run by PEST calculates a prediction of interest, as well as model-based counterparts to members of the calibration dataset. PEST is directed to read that prediction. This prediction can then be

included in the calibration dataset along with the historical observations of system state which are the normal constituents of the calibration dataset. If this prediction is given a relatively high weight, then PEST is effectively instructed to "make the prediction happen". In the decision-making context the prediction may be that of an undesirable consequence of a certain management scenario. If PEST is able to find a reasonable parameter set that allows this unwanted prediction to occur, while simultaneously retaining its ability to fit the historical dataset, then the hypothesis that the unwanted event will occur cannot be rejected. See section 8.5 of Doherty (2015) for more details.

Tikhonov regularisation plays an important role in model-based hypothesis testing of this kind. It can accommodate a large number of parameters. Hence predictive possibilities are not closed through failure to represent parameterisation detail on which the possible occurrence of a unwanted event may depend. Furthermore, the regularisation process can specify that the parameter set which gives rise to an unwanted event must depart to the smallest extent possible from either a calibrated parameter set, or an expected-value parameter set whose values are an expression of expert knowledge. Thus a bad-event hypothesis is not falsely rejected on the basis of an unreasonable parameter field. If desired, the ASSESSPAR utility described in part II of this manual can be used for quantitative assessment of parameter reasonableness.

PEST assists the user in implementing this hypothesis-testing process if the PEST control file cites an observation group named "predict", and if that group contains just a single observation (which can have any name). Under these circumstances PEST prints out the model-calculated number corresponding to that observation every time it calculates a new parameter update vector. Thus whenever it displays a new objective function it also writes the line

```
prediction = x
```

where *x* is the model-generated value corresponding to the sole observation belonging to the observation group "predict".

PEST's use in model-based hypothesis testing can be further formalized if it is run in "pareto" mode. See section 13 of this manual for full details.

9. Tikhonov Regularisation

9.1 General

9.1.1 Advantages of Highly Parameterized Inversion

As is extensively discussed by Doherty (2015), all environmental model calibration requires regularisation of one form or another. In its broadest sense, the term "regularisation" describes the means through which a unique solution is obtained to an inverse problem where the calibration dataset lacks the information to support uniqueness. If PEST is run in "estimation" mode, then pre-calibration, manual regularisation must be undertaken by the modeller to achieve this goal. By running PEST in "regularisation" mode, possibly supported by singular value decomposition or LSQR as numerical solution devices, regularisation is implemented mathematically. This is generally a better alternative than manual regularisation. Advantages of this approach include the following.

- If properly implemented, mathematically regularised inversion promulgates a solution to the inverse problem which is of minimum error variance, and which supports model predictions of minimum error variance. These then form optimal starting points for post-calibration parameter and predictive uncertainty analysis.
- The inversion process is numerically stable. It does not founder for want of an invertible matrix; the inverse problem is formulated in a way that guarantees matrix invertibility.
- Regularised inversion is easy. It does not require that the user examine calibration outcomes and then decide whether the calibration process should be repeated with certain parameters held fixed. Instead a philosophy of "if in doubt, estimate it" can prevail.
- In a two- or three-dimensional model domain, the use of many parameters, supported by a proper regularisation scheme, lets heterogeneity emerge naturally at those locations where its existence is supported by information contained within the calibration dataset, at the same time as it suppresses its emergence where not supported by the data. This is a far better mechanism for calibrating spatial models than restricting expressions of spatial heterogeneity to possibly inappropriate constructs such as zones of assumed piecewise constancy whose design precedes the calibration process and is separate from it.
- Most natural systems are complex and heterogeneous. The use of many parameters allows better representation of system heterogeneity than parsimonious parameterisation. It is precisely because parameterisation detail is unlikely to be uniquely estimable that it must be represented when exploring predictive uncertainty; it is the parameters that cannot be estimated, rather than those which can, which are critical to this endeavour. Regularised inversion easily accommodates parameter nonuniqueness. Post-calibration predictive uncertainty analysis requires exploration of the effects of parameter nonuniqueness if it is to have integrity.

See Doherty (2015) for a full discussion of the advantages of highly parameterized inversion.

9.1.2 Tikhonov Regularisation

As explained in detail by Doherty (2015), appropriately formulated Tikhonov regularisation

provides a "fall-back position" for some or all model parameters. This describes a condition that, according to expert knowledge, they should satisfy unless there is information in the calibration dataset to the contrary. This fall-back condition can be formulated as preferred values for some or all parameters, homogeneity, piecewise homogeneity, minimum curvature of parameter fields, combinations of these, or other conditions altogether.

As well as expressing a preferred parameter condition, Tikhonov regularisation should explicitly or implicitly express the way in which departures from this condition can arise. This can be done through a covariance matrix. Alternatively, it can be done through mechanisms such as "sharpness filters" which penalize the emergence of heterogeneity unless it has sharp edges. Another possibility is to formulate linear and nonlinear "heterogeneity penalty functions" which reward the emergence of heterogeneity at certain locations and in certain directions, and penalise its emergence at other locations and in other directions. In short, Tikhonov regularisation should comprise a mathematical formulation of expert knowledge that allows the inversion process to "join the dots" between the limited parameterisation detail that can be estimated on the basis of the calibration dataset in such a way that the parameter field which emerges from the Tikhonov-regularised inversion process is as compatible with expert knowledge as possible, while allowing the model to replicate field measurements.

Ideally, all parameters involved in the inversion process should feature in Tikhonov regularisation. This ensures that they are all provided with both a fall-back position, and with an optimal means of departing from that fall-back position.

9.1.3 Measurement and Regularization Objective Functions

When PEST is run in "regularisation" mode it defines two objective functions instead of one. The first is the measurement objective function, designated as Φ_m. This is similar to the traditional objective function that PEST uses when run in "estimation" mode in that it constitutes a weighted least squares measure of the discrepancies between field measurements and their model-generated counterparts. The second is the regularisation objective function, designated as Φ_r. This constitutes a weighted least squares measure of the discrepancies between parameters and their preferred conditions.

As stated above, preferred parameter conditions can be simple or complex. The simplest form of regularisation is to assign every estimated parameter a preferred value (normally equal to its initial value). This is easily accomplished using the ADDREG1 utility described in part II of this manual. A series of prior information equations encapsulate this form of regularisation, with one prior information equation pertaining to each parameter. Each prior information equation is then of the following form.

```
pi1 1.0 * par1 = 10.0 1.0 regul1
```

or (if a parameter is log-transformed)

```
pi1 1.0 * log(par1) = 1.0 1.0 regul1
```

To the extent that the "par1" parameter which features in the above equations differs from 10.0, a non-zero residual occurs, and the regularisation objective function itself becomes non-zero.

Prior information can be used to express differences between parameter values or, if parameters are log-transformed, quotients between parameter values. For example

```
pi2 1.0 * par1 - 1.0 * par2 = 0.0 1.0 regul2
```

or (if "par1" and "par2" are log-transformed)

```
pi2 1.0 * log(par1) - 1.0 * log(par2) = 0.0 1.0 regul2
```

To the extent that "par1" differs in value from "par2", these equations acquire a non-zero residual. The regularisation objective function is then non-zero.

It is not necessary that Tikhonov constraints be expressed as prior information equations. They can be calculated by the model rather than by PEST and thus be "observations". Thus they do not need to be linear. Nor do they even need to be comprised of simple equality and difference relationships between parameters; however, of course, they must have some sensitivity to parameters and thus must all feature one or more parameters in some way.

In short, regularisation observations and prior information equations are just like other observations and prior information equations. They have an observed value and a model-calculated value. The difference between these constitutes a residual. Each is assigned a weight. Collectively a group of them can be assigned a covariance matrix. However the distinguishing feature of all of them is that they must be assigned to an observation group whose name begins with "regul". The regularisation objective function is simply the component of the total objective function that is summed over all observations and prior information equations that belong to observation groups whose names begin with "regul". In contrast, the measurement objective function is that component of the total objective function that is summed over all observations and prior information equations that belong to observation groups whose names do not begin with "regul".

9.1.4 Target Measurement Objective Function

Over-fitting is easy, but far from desirable, where many parameters are being estimated. As is explained by Doherty (2015), this can lead to amplification of measurement/structural noise, parameter values which are at odds with expert knowledge, and serious departures from a solution to the inverse problem which is of minimum error variance.

PEST's implementation of Tikhonov regularisation avoids this problem by requesting that the user provide a so-called "target measurement objective function", denoted herein as Φ^t_m. This is the variable PHIMLIM that appears in the "regularisation" section of the PEST control file. The regularised inversion problem is then formulated as a constrained minimisation problem. Specifically, PEST is tasked with minimising the regularisation objective function subject to the constraint that the measurement objective function is equal to its user-supplied target. If PEST cannot lower the measurement objective function to this target value then it lowers it as much as it can.

In solving this constrained minimisation problem, PEST's "lever" is a global multiplier that it applies to all weights that are ascribed to regularisation observations and prior information equations. This multiplier (also referred to as the "regularisation weight factor" herein and in PEST output files) is, in fact, related to the so-called Lagrange multiplier through which solution of the constrained minimisation problem is achieved. As will be discussed below, if requested by the user, PEST has the ability to vary this multiplier between different regularisation groups (i.e. different observation groups whose names begin with "regul" but which end in other letters). However inter-regularisation-group weights adjustment takes place before the global regularisation weight multiplier is applied – see the discussion of the IREGADJ variable below. (Note that this multiplier is applied not only to weights associated with regularisation observations and prior information equations, but to the inverse square root of any covariance matrices that are used instead of weights for groups of regularisation

observations and prior information equations.)

During each iteration of the regularised inversion process PEST minimises a total objective function defined as

$$\Phi = \Phi_m + \mu^2\Phi_r \qquad (9.1.1)$$

where Φ_m and Φ_r are defined as above, and μ is the regularisation weight factor. Its value is calculated anew by PEST during every iteration of the inversion process on the assumption of linear model behaviour. Its value is such that minimisation of the total objective function defined through equation 9.1.1 should result in Φ_m being equal to Φ^t_m.

9.1.5 Solution Mechanism

Theoretically, the use of Tikhonov regularisation not only promulgates a solution to an ill-posed inverse problem which is of minimum error variance. It also brings numerical stability to the solution procedure. It achieves the first of these aims by steering the inverse problem solution procedure in a direction that pays maximum respect to expert knowledge. It achieves the second of these aims by making every parameter a sensitive parameter – if not to observations comprising the calibration dataset, then to regularisation observations which encapsulate expert knowledge.

Unfortunately, however, the use of Tikhonov regularisation alone does not always guarantee unconditional numerical stability. If the target measurement objective function is set very low, then PEST may have no alternative but to calculate a low value for the regularisation weight factor in order to respect the user's desire to fit the data well. For parameters which are sensitive to calibration data this is not a problem. For those which are not, the stabilizing effects of complimentary expert knowledge are lost as the global regularisation weight factor is reduced. Matrices which PEST must invert may then become singular. The inversion process may stall.

This problem can be circumvented to some extent by automatically adjusting weight factor relatively between different regularisation groups. This can be achieved through use of the IREGADJ variable in the "regularisation" section of the PEST control file. However in many circumstances this, alone, cannot guarantee unconditional numerical stability of the inversion process.

Fortunately numerical stability is easily guaranteed if singular value decomposition or LSQR is used for numerical solution of the Tikhonov-regularised inverse problem. If using singular value decomposition, set the MAXSING variable to the number of adjustable parameters (or higher) and EIGTHRESH to 5E-7. Thus singular value truncation takes place only where numerical stability is threatened; meanwhile Tikhonov regularisation takes care of parameter sensibility. If more than two or three thousand parameters must be estimated, then use LSQR instead of singular value decomposition as a solution mechanism for the inverse problem as its speed does not deteriorate as parameter numbers rise, unlike that of singular value decomposition.

9.1.6 Termination Criteria

When running in "regularisation" mode, PEST will cease execution if the following conditions are met.

1. NOPTMAX iterations have elapsed.
2. The measurement objective function falls below the user-supplied target measurement

objective function and the REGCONTINUE regularisation variable is missing or set to "nocontinue".

3. The measurement objective function falls below the value of the optional PHISTOPTHRESH variable supplied in the "control data" section of the PEST control file, regardless of the setting of the REGCONTINUE variable.
4. Parameter changes between iterations are minimal as assessed through reference to the RELPARSTP and NRELPAR variables provided in the "control data" section of the PEST control file.
5. The measurement objective function is greater than PHIMLIM but is falling very slowly as assessed through reference to the PHIREDSTP, NPHISTP and NPHINORED variables provided in the "control data" section of the PEST control file.
6. The measurement objective function is lower than PHIMLIM, REGCONTINUE has been set to "continue" and the regularisation objective function is falling very slowly as assessed through reference to the PHIREDSTP, NPHISTP and NPHINORED variables provided in the "control data" section of the PEST control file.

In practice, it is often best for a user to terminate PEST execution him/herself, especially during an early calibration attempt of a new model where the target measurement objective function may be set very low in an attempt to find out just how good a fit with the calibration dataset PEST can actually achieve.

9.2 Regularisation and the PEST Control File

9.2.1 Setting the Mode

To run PEST in "regularisation" mode, the variable PESTMODE on the second line of the "control data" section of the PEST control file must be set to "regularisation".

9.2.2 Observation Groups

As has already been discussed, when working in "regularisation" mode, observations and/or prior information must be assigned to at least two different observation groups; the name of one of these groups must begin with "regul". Observations and/or prior information items belonging to a group whose name begins with "regul" comprise "regularisation observations"; observations and/or prior information equations belonging to other groups comprise "measurement observations". Weights must be assigned to individual observations and prior information equations of each of these types in the normal manner; alternatively covariance matrices can be assigned to groups of observations or prior information equations of either type. Weights assigned to regularisation observations and/or regularisation prior information equations are multiplied internally by the regularisation weight factor prior to formulation of the total objective function during each iteration of the inversion process. Equivalently, covariance matrices assigned to groups of regularisation observations and prior information equations are divided by the square of the regularisation weight factor.

9.2.3 Regularisation Section

The "regularisation" section of the PEST control file should follow all other sections, including the "prior information" section that may express preferred parameter conditions. Specifications of this section are provided in figure 9.1; as usual, variables enclosed by square brackets are optional. An example is shown in figure 9.2.

```
* regularisation
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE]
WFINIT  WFMIN  WFMAX  [LINREG] [REGCONTINUE]
WFFAC  WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]
```

Figure 9.1 Specifications of the "regularisation" section of the PEST control file.

```
* regularisation
  125.0  130.0  0.1000000
 1.0   1.0e-10    1.0e10
 1.3   1.0e-2     1
```

Figure 9.2 Example of the "regularisation" section of a PEST control file.

The role of each of the variables depicted in figure 9.1 is now discussed in detail.

PHIMLIM

This is the target measurement objective function Φ_m^l. This is the measurement objective function that PEST "aims for" while keeping the regularisation objective function as low as possible. During every iteration of the inversion process PEST calculates a regularisation weight factor that achieves this balance.

If, during any iteration of the inversion process, the measurement objective function falls below PHIMLIM, PEST will cease execution, satisfied that the goal of the inversion process has been achieved. However, with an appropriate setting of the REGCONTINUE variable (see below), it can be instructed to continue with further iterations until the regularisation objective function is as low as it can be subject to the constraint that the measurement objective function is at or below PHIMLIM. Normally this will result in raising the measurement objective function back up to the PHIMLIM threshold.

When attempting calibration of a model for the first time, you may not know how good a fit can be achieved between model outputs and the calibration dataset, and hence what is a suitable setting for PHIMLIM. For these initial runs PHIMLIM can be set very low, for example it can be set to 1E-10 just to see how good a fit can be achieved between model outputs and field measurements. Meanwhile, the FRACPHIM control variable (see below) can be set to a value such as 0.1; this prevents regularisation constraints on parameter values from being totally ignored on the way to achieving the best possible fit with the data. If it turns out that the fit between model outcomes and the calibration dataset is poor, this may indicate that the model needs refinement; by seeking the best possible fit with the calibration dataset you get to find this out early in the calibration process. However if the fit is good, then PHIMLIM can be set to a value 5 percent to 10 percent higher than the best measurement objective function achieved during this PEST run prior to running PEST again. Parameter values achieved during the next PEST run should be far more pleasing than those achieved during the first run; it is normally in achieving the last 5 percent to 10 percent reduction in the measurement objective function that parameters are assigned values that violate sensibility constraints as over-fitting occurs.

If there is not enough time to undertake more than one PEST run, and this run is undertaken with PHIMLIM set very low, then parameter values calculated during early iterations of the inversion process when the measurement objective function was not quite so low as to constitute over-fitting may be adopted as the "calibrated parameter set". These can be retrieved if the PARSAVEITN variable in the "control data" section of the PEST control file is set to "parsaveitn".

PHIMACCEPT

During each iteration, just after it has linearized the problem through calculating the Jacobian matrix, and just before it begins calculation of the parameter upgrade vector, PEST calculates the optimal value of the regularisation weight factor for that iteration. This is the value which, under the linearity assumption encapsulated in the Jacobian matrix, results in a parameter upgrade vector for which the measurement objective function Φ_m is equal to PHIMLIM (i.e. Φ_m^l). However, due to the approximate nature of the linearity assumption, PEST may not be able to lower Φ_m to Φ_m^l on that iteration in spite of the fact that it uses a number of different values for the Marquardt lambda in attempting to do so. If it cannot lower Φ_m to an acceptable level, it simply accepts the upgraded parameters, proceeds to the next iteration and tries again. However if it does succeed in lowering Φ_m to an acceptable level, or if it has succeeded in doing this on previous iterations, then PEST slightly alters its philosophy of choosing new Marquardt lambdas, in that it now attempts to lower Φ_r while maintaining Φ_m below this acceptable level. This acceptable level is PHIMACCEPT; it should be set slightly higher than PHIMLIM (i.e. Φ_m^l) in order to give PEST some "room to move" in its attempts to lower Φ_r while keeping Φ_m below, or close to, Φ_m^l. It needs this "room to move" because of the fact that it bases its calculations on a linearity assumption that is only approximately satisfied.

Normally PHIMACCEPT should be about 5 percent to 10 percent greater than PHIMLIM. However if PEST is performing well, you may wish to make it closer to PHIMLIM than this. In choosing the best parameter set at any stage of the inversion process (for recording in the parameter value file) PEST looks at all parameter sets for which it has carried out model runs up to that point in the process. If any of these runs have resulted in a measurement objective function less than PHIMACCEPT, it then searches from among these runs for the parameter set which gave rise to the lowest regularisation objective function. If PHIMACCEPT is set too close to PHIMLIM, PEST's selection of the best parameter set may be somewhat restricted, for there may be some parameter sets for which Φ_m is just above PHIMACCEPT but for which Φ_r is quite low. Alternatively, if PHIMACCEPT is set too large, then PEST might not try hard enough to reduce Φ_m to Φ_m^l, preferring instead to work within the weaker constraint set by PHIMACCEPT. When working in "regularisation" mode, PEST prints out Φ_r and Φ_m for every parameter upgrade attempt. It will be apparent from this information whether PHIMACCEPT has been set correctly.

In general, set PHIMACCEPT 5 percent higher than PHIMLIM. This works fine in the vast majority of cases.

FRACPHIM

The FRACPHIM variable allows you to set PHIMLIM very low (possibly lower than is achievable), but still retain the benefits of regularisation.

If FRACPHIM is provided with a value of zero or less (or if this variable is absent from the PEST control file), then it has no effect on the inversion process. However if FRACPHIM is provided with a value of between 0.0 and 1.0 (values of 1.0 or greater are illegal), then PEST calculates a new value for PHIMLIM at the beginning of each iteration of the inversion process. This value is calculated as the current value of the measurement objective function times FRACPHIM, or the user-supplied value of PHIMLIM, whichever is higher. Thus PEST will always "aim for" a measurement objective function that is lower than the current one. However it does not pursue a target that is so low as to require the complete abandonment of

regularisation.

As well as adjusting the value of PHIMLIM during every iteration, PEST also adjusts the value of PHIMACCEPT. This adjustment is made such that, during every iteration, the ratio of PHIMACCEPT to PHIMLIM is the same as that supplied in the PEST control file.

The recommended value for FRACPHIM is 0.1.

MEMSAVE

The optional MEMSAVE variable can be used to implement memory conservation features that may assist PEST in very highly parameterized cases. MEMSAVE is a character variable which must be supplied as either "memsave" or "nomemsave". If supplied, it follows the optional FRACPHIM variable. If FRACPHIM is omitted MEMSAVE follows PHIMACCEPT. See section 15.7 of this manual for further details.

WFINIT

This is the initial regularisation weight factor. During every iteration of the inversion process PEST calculates a suitable regularisation weight factor to employ during that iteration using an iterative, numerical solution procedure; its initial value when implementing this procedure for the first iteration is WFINIT. If there are many adjustable parameters, calculation of the regularisation weight factor for the first iteration can be time-consuming if WFINIT is far from optimal. Hence if you have any idea of what the weight factor should be (for example from a previous PEST run), then you should provide WFINIT with this value. Otherwise simply set it to 1.0.

WFMIN, WFMAX

These are the minimum and maximum permissible values that the regularisation weight factor is allowed to take. Normally settings of 1E-10 and 1E10 are suitable; settings of 1E-15 and 1E15 are normally fine as well. If PEST wishes to transgress these limits it will notify you of this. This normally indicates that regularisation constraints are too weak to ensure inverse problem uniqueness (if the upper weight factor limit is encountered), or that measurement weights are too low (if the lower weight factor limit is encountered).

WFFAC, WFTOL

When PEST calculates the appropriate regularisation weight factor to use during any iteration of the inversion process, it uses an iterative procedure which begins at the value of the regularisation weight factor calculated during the previous iteration; for the first iteration it uses WFINIT to start the procedure. In the process of finding the weight factor which, under the linearity assumption used in its calculation, will result in a measurement objective function (i.e. Φ_m) of PHIMLIM (i.e. Φ_m^l), PEST first travels along a path of progressively increasing or decreasing weight factor (it decides which one of these alternatives to explore on the basis of the value of the current measurement objective function with respect to PHIMLIM). In undertaking this exploration, it either multiplies or divides the weight factor by WFFAC; it continues to do this until it has found two successive weight factors which lie on either side of the optimal weight factor for that iteration. Once it has done this, it uses Newton's method to calculate the optimal weight factor, through a series of successive approximations. When two subsequent weight factors calculated in this way differ from each other by no more than a relative amount of WFTOL, the optimal weight factor is deemed to have been calculated.

Experience has shown that a suitable value for WFFAC is about 1.3; it must be greater than 1.0. WFTOL is best set at somewhere between 1E-3 and 1E-2. However if there are many adjustable parameters and PEST consumes a large amount of time in determining the optimal weight factor, a tolerance of somewhat higher than 1E-2 may prove suitable.

LINREG

As was discussed above, regularisation constraints can be supplied through observations, through prior information, or through both of these mechanisms. Prior information relationships are always linear. Regularisation constraints supplied as observations (for which the current values of pertinent relationships are calculated by the model), can be linear or nonlinear; in either case, derivatives of these relationships with respect to adjustable parameters are re-evaluated by PEST during each iteration.

If regularisation information is entirely linear, there are many matrix operations carried out as part of PEST's regularisation functionality which do not need to be repeated from iteration to iteration. If repetition of these calculations can be avoided in parameter estimation contexts involving many regularisation constraints, significant gains in efficiency can be made. The user can inform PEST that all regularisation constraints are linear through the optional LINREG control variable. LINREG should be supplied as either "linreg" or "nonlinreg".

If a value for LINREG is not supplied in a PEST control file, the default value of "nonlinreg" is employed, unless all regularisation constraints are supplied as prior information, in which case the default value of "linreg" is used. The location of the optional LINREG variable on the third line of the "regularisation" section of the PEST control file is interchangeable with that of the optional REGCONTINUE variable. (Both of these are text variables, so their placement is easily recognized by their value.) If either is present, it must follow the WFMAX variable.

It is very important to take account of the transformation status of parameters when assigning a value to LINREG. Regularisation relationships must be linear with respect to the log of parameter values for those parameters that are designated as log-transformed in the "parameter data" section of the PEST control file. They must be linear with respect to parameter values themselves for those parameters that are not log-transformed.

REGCONTINUE

Under normal circumstances, when working in "regularization" mode, PEST ceases execution immediately if the measurement objective function falls below its user-supplied target value of PHIMLIM. Therefore it does not undertake further iterations in an attempt to lower the regularisation objective function any further in order to maximise the extent to which parameters adhere to the preferred condition that is encapsulated in regularisation constraints.

There are some circumstances, however, where minimisation of the regularisation objective function is just as important as allowing the measurement objective function to reach PHIMLIM. In these circumstances, PEST should continue with the parameter estimation process until some other convergence criterion is met (for example that parameters cease to change noticeably between iterations, or that neither the regularisation nor measurement objective function changes noticeably between iterations). This can be achieved through setting the REGCONTINUE variable to "continue". This text variable must follow WFMAX on the third line of the "regularisation" section of the PEST control file. Its position is interchangeable with that of the optional LINREG variable.

If present, REGCONTINUE must be supplied as either "continue" or "nocontinue". If absent, it is assumed to be "nocontinue"; in this case PEST ceases execution as soon as the measurement objective function falls below PHIMLIM. If it is set to "continue" however, PEST will continue iterating, trying to improve the regularisation objective function, until some other convergence criterion is met.

IREGADJ and Related Variables

If the optional IREGADJ variable is missing from the "regularisation" section of the PEST control file, or is provided with a value of zero, then no inter-regularisation-group weights adjustment takes place. Thus the global weight factor calculated by PEST, and updated during every iteration of the inversion process, is applied uniformly to all regularisation observations and prior information equations. Relativity of regularisation weighting as provided by the user is thus maintained.

However if IREGADJ is provided with a non-zero value, this instructs PEST to vary relatively of regularisation weighting in ways that are discussed in the next section. Various strategies can be implemented according to the setting of this variable. Though differing widely in their details, they all attempt to strengthen regularisation constraints on parameters whose values are relatively uninformed by the calibration dataset while loosening constraints on those of which the calibration dataset is informative.

In most cases an IREGADJ setting of 1 works well. This and other options are now discussed in more detail.

9.3 Regularisation Weight Relativity Adjustment

9.3.1 Overview

Use of Tikhonov regularisation in the inversion process often results in parameter fields that "look good" from a geological or other perspective. This will be especially the case if Tikhonov constraints have been formulated in such a way that the "default parameter condition" that they express represents an expert's encapsulation of pre-calibration parameterisation maximum likelihood. However not only should the preferred condition express parameterisation reasonableness; in addition to this, weights and/or covariance matrices applied to regularisation constraints should be such that violation of these constraints in order to allow model outputs to fit field data takes place in a reasonable, or "least-unlikelihood" fashion. In the spatial modelling context (for example groundwater or reservoir models), this will result in the introduction of heterogeneity that is of a scale and disposition that "makes sense" in the geological setting under consideration.

A practical problem that often arises when using Tikhonov regularisation is that neither the "default condition" of real-world parameters, nor the covariance matrix of spatial or temporal parameter heterogeneity that describes their potential variability, is known (or even very applicable). Another problem is that, as the inversion process advances and PEST pursues a good fit with field data in areas of high spatial data density where there may be strong evidence for the existence of parameterisation heterogeneity, it "let's go" of Tikhonov regularisation constraints in order to achieve these fits, and to therefore express the heterogeneity which they suggest. "Letting go" takes place through reduction of the regularisation weight factor. However an unfortunate consequence of "letting go" is that numerical instability of the inversion process may be incurred, for the consequential lack of constraints on parameters in data-poor parts of the model domain may result in

nonuniqueness of estimation of parameters in those areas. Matrix condition numbers then rise, and the consequential numerical instability will almost certainly hamper further progress of the inversion process.

This problem can be partially overcome through applying differential weighting to regularisation constraints, with regularisation weights being weaker on parameters (and/or parameter combinations) that are relatively estimable on the basis of the current calibration dataset, while being stronger on constraints that pertain to parameters (and/or parameter combinations) that are relatively inestimable. Thus the information content of the calibration dataset can be transferred to parameters (and parameter combinations) of which this dataset is informative, without the need to relax application of default conditions on parameters (and parameter combinations) for which information in the calibration dataset is weak or absent.

PEST is given license to adopt differential weighting for regularisation prior information equations and regularisation observations through use of the IREGADJ regularisation control variable. Differential weighting is computed and applied on a group-by-group basis if IREGADJ is set 1, 2 or 3. However if it is set to 4 or 5, regularisation weights adjustment takes place on an item-by-item basis, where an "item" is an individual regularisation prior information equation, or an individual regularisation observation.

9.3.2 IREGADJ Settings of 1 and 2

As has already been discussed, different regularisation observations and prior information equations can be placed into different regularisation groups. The names of each of these groups must begin with the characters "regul". As for any observation group, the total name must be 12 characters or less in length. Ideally, regularisation constraints applied to different types of parameters should be placed into different groups. Thus, for example, if a groundwater or reservoir model is multi-layered, then regularisation constraints applied to layer 1 permeabilities should be placed into a different regularisation group from regularisation constraints applied to layer 2 permeabilities. In any one layer, regularisation constraints on permeabilities should belong to a different group from regularisation constraints on storage coefficients.

One beneficial outcome of placing regularisation constraints into different groups is that PEST, in accordance with its normal reporting protocols, will record the contribution made to the total objective function by each such group. Another benefit is that PEST is then able to implement automatic inter-group weights adjustment if directed to do so through appropriate settings of the IREGADJ control variable.

For a particular observation or prior information equation, the "composite observation sensitivity" is defined using equation 5.3.2; this is now repeated.

$$cso_j = \frac{\left[\mathbf{Q}^{1/2}\mathbf{J}\mathbf{J}^t\mathbf{Q}^{1/2}\right]_{jj}^{1/2}}{m} \qquad (9.3.1)$$

As has been previously discussed, the composite sensitivity of observation *j* is the magnitude of the *j'*th row of the Jacobian matrix multiplied by the weight associated with that observation; this magnitude is then divided by the number of adjustable parameters. It is thus a measure of the sensitivity of that observation to all parameters involved in the parameter estimation process. The total composite observation sensitivity of a particular observation group can be calculated by summing the composite sensitivities of all observations or prior information equations comprising that group.

If IREGADJ is set to 1, PEST multiplies the weights pertaining to all members of each regularisation group by a group-specific factor. This factor is chosen so that, after this operation has been performed, the total composite sensitivities of all regularisation groups are the same. It is important to note, however, that in performing these calculations, relative weighting within each observation group remains unchanged.

If IREGADJ is set to 2, PEST simply sums the weights pertaining to all members of each regularisation group and then calculates a relative weight factor for each group such that the sum of these weights is the same for all regularisation groups. Once again, relative within-group weighting is preserved.

9.3.3 IREGADJ Setting of 3

An IREGADJ setting of 3 combines the benefits of an IREGADJ setting of 1 with a user's advice on which parameters should be allowed to undergo greater adjustment than others through the inversion process. This is helpful in situations where adjustment of either of two sets of parameters may promulgate a better fit between field measurements and corresponding model outputs.

If IREGADJ is set to 3, PEST undertakes the same calculations for the purpose of relative group weight factor calculation as those which it undertakes when IREGADJ is set to 1. However it then undertakes a "final adjustment" of regularisation weights by multiplying them all by user-supplied regularisation weights. Thus if, for example, a user supplies weights for group "regul1" which are twice those for "regul2", PEST will multiply all weights within group "regul1" by a factor of 2 relative to those in group "regul2" *after* having calculated regularisation weights for these groups using the procedure that is employed when IREGADJ is set to 1. All regularisation weights are then multiplied by the global, iteration-specific, PEST-calculated regularisation weight factor before parameter estimates are then upgraded.

Setting IREGADJ to 3 has the potential to be very useful where there are many regularisation groups. In such circumstances it is difficult to determine a "balanced" set of inter-regularisation group weight factors yourself; the result may be poor PEST performance as regularisation constraints fail to compensate for data inadequacy. Although PEST may be able to overcome this problem with an IREGADJ setting of 1, it may nevertheless be your desire that regularisation constraints for some parameters be enforced more strongly than for others. In these circumstances you can supply an initial set of weights that reflect this desire. PEST then accommodates this desire as it tries to make all regularisation constraints visible to the inversion process.

It is apparent from the above discussion that it would be unwise to supply regularisation weights which are markedly different from group to group when IREGADJ is set to 3, for the benefits of composite-sensitivity-based weights adjustment will be lost if relative inter-group weighting is varied too much from the "balanced" weighting administered by PEST. When IREGADJ is set to 1, PEST's internal weights adjustment procedure automatically take into account the population of each regularisation group, and the composite sensitivity of each of the observations or prior information equations comprising each regularisation constraint. When IREGADJ is set to 3, you have an opportunity to "fine-tune" the balanced weighting regime introduced by PEST in order to reflect your desire for certain regularisation constraints to be enforced more than others. However if such tuning prevents stable solution of the inverse problem, then the desire for stronger enforcement of one set of constraints over another should be abandoned; either uniform regularisation weights should then supplied, or

IREGADJ should be set to 1.

9.3.4 IREGADJ Settings of 4 and 5

Theory

IREGADJ settings of 4 and 5 support so-called "subspace-enhanced" Tikhonov regularisation in which singular value decomposition is employed in order to promulgate application of tighter regularisation constraints on parameters of which the calibration dataset is less informative.

Let **J** represent the Jacobian matrix for the current inverse problem, and let **k** represent parameters that are featured in this problem. Let **Q** represent the observation weight matrix. Through singular value decomposition of the weighted Jacobian matrix we have

$$\mathbf{Q}^{1/2}\mathbf{J} = \mathbf{USV}^{t} \tag{9.3.2}$$

where, as usual, the columns of **V** comprise orthogonal unit vectors which collectively span parameter space. On the basis of a suitable singular value threshold, **V** can be subdivided into submatrices $\mathbf{V}_1$ and $\mathbf{V}_2$ such that

$$\mathbf{Q}^{1/2}\mathbf{J} = \mathbf{U}\begin{bmatrix}\mathbf{S}_1 & \mathbf{S}_2\end{bmatrix}\begin{bmatrix}\mathbf{V}_1^{t} \\ \mathbf{V}_2^{t}\end{bmatrix} \tag{9.3.3}$$

Ideally, the threshold should be chosen such that $\mathbf{V}_1$ spans the "calibration solution space" while $\mathbf{V}_2$ spans its orthogonal complement, the "calibration null space"; see Doherty (2015) for a full discussion of these concepts.

Let the vector **y** represent the sensitivities of a regularisation prior information equation, or regularisation observation, to the parameters **k**. If **y** has a large projection onto the calibration solution space, then the model-generated outcome of the prior information equation or observation is in fact well determined on the basis of information resident in the calibration dataset. If the "observed" value of this equation or observation contradicts its value as generated by the calibrated model, then PEST should reduce the weight applied to this particular regularisation constraint. If IREGADJ is set to zero it will thereby reduce the weight applied to all regularisation constraints. This may then reduce the beneficial effects of regularisation on relatively inestimable parameters. At worst it will lead to near-singularity of matrices that PEST must invert, and thence to numerical instability. (However, as stated above, numerical instability can be forestalled if singular value decomposition or LSQR is used to solve the inversion equations.)

On the other hand, if the vector **y** has a small projection onto the calibration solution space, then the calibration dataset provides no information that contradicts the user-supplied value of this constraint, irrespective of its value. Hence this constraint should be tightly applied, with its weight maintained or increased in order to ensure that this occurs.

IREGADJ Settings

Subspace enhancement of Tikhonov regularisation is implemented by PEST in one of two ways, depending on whether the IREGADJ variable is set to 4 or 5. If IREGADJ is set to 4, PEST adopts the following procedure in assigning weights to individual regularisation observations and prior information equations.

1. PEST formulates $\mathbf{Q}^{1/2}\mathbf{J}$ on the basis of the current Jacobian matrix **J**, but from which

all regularisation constraints are omitted. Thus **J** comprises only the calibration dataset.

2. Singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ is undertaken – see equation 9.3.2.
3. From the current Jacobian matrix PEST obtains the sensitivity vector **y** pertaining to each regularisation observation and each regularisation prior information equation included in the current inversion process.
4. For each such **y**, for example $\mathbf{y}_j$, the direction cosine

$$\cos(\theta_{ji}) = \frac{\mathbf{y}_j^t \mathbf{v}_i}{\sqrt{\mathbf{y}_j^t \mathbf{y}_j}} \tag{9.3.4}$$

is computed for each column $\mathbf{v}_i$ of **V**.

5. The weight for regularisation constraint *j* is calculated as

$$w_j = \sum_i \frac{\cos(\theta_{ji})}{s_i} \tag{9.3.5}$$

where s_i is the singular value associated with the unit vector $\mathbf{v}_i$.

This strategy results in small weights being assigned to regularisation constraints that are informed by the data (and therefore run the risk of contradicting that data) and large weights being assigned to regularisation constraints that are not informed by the data, and hence which are needed for stable estimation of the parameters that they cite.

Application of equation 9.3.5 can result in weights for regularisation constraints which can vary over many orders of magnitude. Two devices are employed to limit this variability.

First, in calculating w_j through equation 9.3.5, PEST will not employ an s_i value which is less than 10^{-7} of the maximum singular value pertaining to the current $\mathbf{Q}^{1/2}\mathbf{J}$ matrix. Instead, a value of 10^{-7} times the maximum singular value is employed in place of singular values which are less than this. Second, once weights for all regularisation constraints have been computed using equation 9.3.5, the range of regularisation weight magnitudes is compressed such that the ratio of the largest to the smallest resulting weight is equal to a user-specified value. Weights compression can be linear or logarithmic. In the former case, alterations to weight magnitudes are proportional to those magnitudes as calculated using equation 9.3.5. In the latter case, alteration is proportional to the logs of weight magnitudes.

If IREGADJ is set to 5 rather than 4, each regularisation observation and prior information equation is assigned either of two weight values, with the ratio of these values being set by the user. The procedure is as follows.

1. The user selects a singular value ratio for which **V** of equation 9.3.2 is partitioned into $\mathbf{V}_1$ and $\mathbf{V}_2$ of equation 9.3.3.
2. If, for regularisation constraint *j*, the sum of squared direction cosines $\cos(\theta_{ji})$ onto eigenvectors comprising the columns of $\mathbf{V}_1$ is greater than 0.5, the regularisation constraint is assigned the lower of the two user-supplied weights.
3. If the sum of squared direction cosines $\cos(\theta_{ji})$ onto eigenvectors comprising the columns of $\mathbf{V}_1$ is less than 0.5, the regularisation constraint is assigned the greater of the two user-supplied weights.

It must be remembered that, irrespective of whether IREGADJ is set to 4 or 5, regularisation weights are subjected to global weights adjustment during each iteration of the inversion process in accordance with PEST's normal procedure for implementing Tikhonov regularisation based on attainment of a measurement objective function equal to the user-supplied value of PHIMLIM. The above procedures determine relativity of regularisation weighting.

IREGADJ Support Variables

Three variables follow IREGADJ in the "regularisation" section of the PEST control file to govern implementation of subspace-enhanced Tikhonov regularisation. See figure 9.1. If IREGADJ is set to 4, values must be supplied for NOPTREGADJ and REGWEIGHTRAT; if it is set to 5, a value must also be supplied for REGSINGTHRESH.

NOPTREGADJ (an integer) specifies the optimisation interval at which regularisation weights are re-calculated in the manner discussed above. If the number of observations and/or parameters involved in the inversion process is large, undertaking singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix may be a computationally expensive exercise. The user may thus wish to avoid having to do this during every iteration. This can be achieved by setting NOPTREGADJ to a number greater than 1. Thus if, for example, NOPTREGADJ is set to 3, regularisation weights re-calculation takes place during every third iteration. Note that it is always undertaken on the first iteration of the inversion process.

REGWEIGHTRAT (a real number) specifies the ratio of the highest to lowest regularisation weight enforced through the weights compression process described above. A setting of between 10 and 100 is suggested; however this suggestion may change as further experience in the use of subspace-enhanced Tikhonov regularisation is gained.

The REGSINGTHRESH (real) variable is required in the PEST control file only if IREGADJ is set to 5. It defines the singular value ratio which is deemed to separate the calibration solution space from the calibration null space for the purpose of assigning a weight to each regularisation observation and prior information equation in the manner described above. For consistency with the EIGTHRESH variable used in the "singular value decomposition" section of the PEST control file, this ratio actually applies to singular values of the $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix rather than those of the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix (the former are squares of the latter). This being the case, a lower limit of about 1E-7 is suggested for this variable, this corresponding approximately to a suitable level for separation of the solution space from the null space where there is little or no measurement noise associated with the calibration dataset. More commonly, however, this should be set to a much higher value, for example 1E-5. However, as for other suggested values for the control variables discussed herein, this suggestion may be modified with experience.

If IREGADJ is set to 5, the REGWEIGHTRAT variable determines the ratio of null space to solution space regularisation weights. NOPTREGADJ determines the iteration interval at which regularisation weights adjustment takes place.

Experience to Date

Experience to date in using subspace-enhanced Tikhonov regularisation has been mixed. Normally an IREGADJ setting of 1 or 3 works well (1 is sufficient in most cases). Some PEST users have reported good outcomes when setting IREGADJ to 4 however.

9.4 PEST Run-Time and End-of-Run Information

9.4.1 Run-Time Information

As it runs, PEST records information to both the screen and to its run record file. When PEST is run in "estimation" mode, the principal items of interest during each iteration are the value of the objective function at the beginning of the iteration, and new values of the objective function which are computed as PEST tests a series of parameter upgrade vectors calculated on the basis of a number of different Marquardt lambdas. When run in "predictive analysis" mode, the current value of the model prediction is also important.

The situation is slightly different when PEST runs in "regularisation" mode. Because the regularisation weight factor is different from iteration to iteration, the total objective function calculated during one iteration of the inversion process is not directly comparable with that calculated by PEST during the previous iteration. If IREGADJ is set to zero (or missing from the "regularisation" section of the PEST control file) then the measurement and regularisation objective functions are individually comparable from iteration to iteration. However if IREGADJ is set to a value other than zero, PEST has license to adjust inter-regularisation-group weighting relativity. Hence regularisation objective functions are not then comparable from iteration to iteration.

Figure 9.3 shows part of a run record file produced by PEST when operating in "regularisation" mode.

```
OPTIMISATION ITERATION NO.        : 3
   Model calls so far             : 235
   Current regularisation weight factor                    :   5.1287
   Current value of measurement objective function         :   22.272
   Current value of regularisation objective function      :  0.17298
   Note: regularisation objective function is not comparable between
   iterations because of IREGADJ regularisation weights adjustment.

   Starting phi for this iteration                         :   26.822
   Contribution to phi from observation group "obsgp1"     :   2.8077
   Contribution to phi from observation group "obsgp2"     :   19.465
   Contribution to phi from observation group "regul"      :   4.5499

   Re-calculated regularisation weight factor              :   14.941
   New starting objective function for this itn. (ie. phi) :   60.887
   Contribution to phi from observation group "obsgp1"     :   2.8077
   Contribution to phi from observation group "obsgp2"     :   19.465
   Contribution to phi from observation group "regul"      :   38.615

       Lambda =   2.5000     ----->
          Phi =   22.035      (  0.362 of starting phi)
    Meas. fn. =   3.0926
   Regul. fn. =  8.48553E-02

       Lambda =   1.2500     ----->
          Phi =   19.463      (  0.320 of starting phi)
    Meas. fn. =   2.8236
   Regul. fn. =  7.45386E-02

       Lambda =  0.62500     ----->
          Phi =   17.358      (  0.285 of starting phi)
    Meas. fn. =   2.6310
   Regul. fn. =  6.59695E-02
```

```
   No more lambdas: phi is less than 0.3000 of starting phi

      Current parameter values                 Previous parameter values
             .
             .
   Maximum   factor change:  2.633     ["ro6"]
   Maximum relative change:  1.633     ["ro6"]

OPTIMISATION ITERATION NO.        : 3
etc
```

Figure 9.3 Extract from a PEST run record file.

PEST begins each iteration by recording the current value of the regularisation weight factor (as calculated during the previous iteration) and the values of the regularisation and measurement objective functions Φ_r and Φ_m. Note that user-supplied regularisation weights are not multiplied by the current weight factor when calculating the regularisation objective function. (That is why the value of the regularisation objective function is comparable from iteration to iteration if IREGADJ is set to 0.) Of course, no weight factor is used in calculating the measurement objective function.

Next PEST fills the Jacobian matrix. Then, on the basis of the linearity assumption encapsulated in the Jacobian matrix, PEST calculates the optimal value of the regularisation weight factor for the current iteration. Once it has calculated the regularisation weight factor, it can calculate an objective function (i.e. "phi" in figure 9.3) for the current iteration. PEST prints this phi as "the starting objective function for this itn.".

PEST then calculates one or a number of parameter upgrade vectors on the basis of one or a number of different Marquardt lambdas in an attempt to lower the objective function as much as possible. In lowering the total objective function PEST thereby lowers one or both of Φ_r and Φ_m. Marquardt lambdas are selected using a similar procedure to that used when PEST is working in "estimation" mode; however this is modified to account for the presence of two objective functions. For each parameter upgrade vector that it tests, PEST lists the measurement and regularisation objective functions as well as the total objective function calculated on the basis of a model run undertaken with the trial parameters. Note that the sum of the measurement and regularisation objective functions will not equal the total objective function unless the regularisation weight factor is unity.

As described in section 5.3.3 of this manual, in the course of its execution PEST records the composite sensitivities of all adjustable parameters in a parameter sensitivity file. The composite sensitivity of any parameter can be considered as the magnitude of the vector comprising the weighted column of the Jacobian matrix corresponding to that parameter, divided by the number of observations. Where some of the observations taking part in the parameter estimation process are regularisation observations, their weights will change from iteration to iteration in accordance with the current value of the regularisation weight factor. The changing weight factor alters the values of the composite parameter sensitivities. Hence these sensitivities are not directly comparable from iteration to iteration.

9.4.2 Post-Run Information

In contrast to its "estimation" mode protocol, PEST does not record post-calibration covariance and associated matrices either to files named *case.mtt* (where *case* is the filename base of the PEST control file) during the course of the inversion process, or to the run record

file *case.rec* on completion of the inversion process. If required, the PREDUNC7 utility can be used to obtain a posterior covariance matrix. Other utilities described in part II of this manual can be used for other types of post calibration parameter and predictive uncertainty analysis. (It is important to note that information on parameter estimability forthcoming from the covariance and related matrices that PEST calculates when working in "estimation" mode is not required when PEST is run in "regularisation" mode. It is a foregone conclusion that the inverse problem is ill-posed; these matrices are therefore inapplicable at best and incalculable at worst.)

Like the run record file produced by PEST when run in "estimation" mode, the run record file produced as an outcome of a regularisation run contains a listing of residuals and respective weights, together with a brief statistical summary of the residuals pertaining to each observation group. It should be noted that wherever weights are cited, or are used in any statistical calculation in this section of the PEST run record file, the weights pertaining to regularisation observations and prior information equations are multiplied by the optimised regularisation weight factor, i.e. by the regularisation weight factor used in calculation of parameter values that are decreed to be the solution to the inverse problem. Estimated parameter values are listed on the run record file and, of course, in the parameter value file. These parameters may or may not have been calculated during the final PEST iteration; this depends on the behaviour of the measurement and regularisation objective functions in iterations preceding the final iteration.

Similar considerations apply to information recorded in the residuals file (*case.res*), and possibly the rotated residuals file (*case.rsr*), at the end of the inversion process. That is, where weights are used in the calculation of any quantities pertaining to regularisation observations listed in these files, the regularisation weights supplied by the user are multiplied by the optimised regularisation weight factor. If IREGADJ is set to a non-zero value they will also have been subjected to adjustment on a group or individual basis. "Measurement standard deviations" and "natural weights" are not calculated for regularisation residuals, as these have no meaning.

Information recorded in the parameter and observation sensitivity files produced by PEST at the end of its run is also weight-dependent. Weights pertaining to regularisation observations and prior information equations differ from user-supplied weights for these quantities in ways already described.

9.5 Group-Specific Target Objective Functions

9.5.1 General

We conclude this section with an item of PEST functionality which, to the author's knowledge, has not been used a great deal, but which may be of use in some inversion contexts. The method is new and should be used with caution.

As has been extensively described above, when PEST is run in "regularisation" mode, a value must be supplied for the PHIMLIM variable in the "regularisation" section of the PEST control file. This is the target measurement objective function. When run in "regularisation" mode, PEST is asked to minimise the regularisation objective function subject to the constraint that the measurement objective function is equal to PHIMLIM. In practice PHIMLIM may not be attainable, in which case PEST lowers the measurement objective function as far as it can. If it is obtainable, PEST ceases execution as soon as the measurement objective function falls below PHIMLIM. Alternatively, PEST can be asked to

continue the inversion process until the regularisation objective function is as small as it can be while the measurement objective function is simultaneously made as close as possible to its target; this mode of behavior is activated by setting the REGCONTINUE variable to "continue".

Sometimes it can be useful to set the target measurement objective function on an observation group by observation group basis. When this is done, PEST is asked to monitor the performance of the inversion process in lowering each group-based component of the measurement objective function to its group-specific target. If one such target it not being met, PEST raises the weights assigned to members of the pertinent observation group so that the overall objective function penalty incurred through not meeting that target is greater; PEST is thus provided with more incentive to meet it. This, of course, provides no guarantee that the target will actually be met. However it does provide PEST with greater encouragement to meet as yet unmet targets than would otherwise be the case.

When weights are altered, so too does the current value of the measurement objective function, as indeed do target measurement objective functions. However when undertaking observation weights adjustment in the manner just described, PEST ensures that the overall target measurement objective function (which is equal to the sum of group-specific target measurement objective functions expressed in terms of original measurement weights as supplied in the PEST control file) does not change. While reporting the revised current measurement objective function to the screen and to its run record file at all stages of the parameter estimation process, it also reports the measurement objective function, and group-specific components thereof, as calculated from original weights, both during and after the parameter estimation process. The metric by which success of the constrained inversion process is judged is that these individual measurement objective function components approach (or are less than) their group-specific measurement objective function targets (which, as is stated above, are calculated on the basis of original weights as supplied in the PEST control file). In the meantime, if the measurement objective function, as reported on all attempts to upgrade parameters using different Marquardt lambdas, is lower than the initial target (obtained through summation of individual group-specific targets) then so too will be the measurement objective function calculated using initial weights.

The following should be noted.

1. Group-specific measurement objective function targets must be supplied for all non-regularisation observation groups, or for none of them at all.
2. A group-specific measurement objective function target must not be supplied for a regularisation group.
3. If group-specific measurement objective function targets are supplied, values for PHIMLIM and PHIMACCEPT as supplied in the "regularisation" section of the PEST control file are ignored. PHIMACCEPT is internally set to 1.05 times the total measurement objective function target value.
4. If a value is supplied for FRACPHIM in the "regularisation" section of the PEST control file, this is respected. Thus if FRACPHIM times the current objective function is greater than the overall target measurement objective function, the former is accepted as the temporary target for the current iteration.
5. Internal weights re-assignment by PEST will never be such that weights for any observation group are increased by a factor of more than 128 or reduced by a factor of

128 from their original values. The increase/reduction factor per iteration is never greater than 2.0.

6. Unless REGCONTINUE is set to "continue" PEST will cease execution when the total measurement objective function falls below the total target measurement objective function; no account is taken of whether groups-specific measurement objective functions are below their group-specific targets.

Provision of group-specific measurement objective function targets provides no guarantee that these targets will all be simultaneously met. However it can provide some assistance to the inversion process in achieving this result. The user should try, however, to limit the need for measurement weights adjustment by choosing original weights and group-specific targets in a manner that is considered optimal prior to commencement of the inversion process. For example, if it is apparent to the user that a group-specific measurement objective function target is very easily achieved, then a low value for that target should be provided to PEST. This reduces unnecessary "target headroom", thus allowing PEST to focus on reducing other components of the measurement objective function to their desired target levels.

9.5.2 Implementation

Group-specific measurement objective function targets must be supplied in the "observation groups" section of the PEST control file. Each such target must be placed immediately following the name of a non-regularisation observation group. If a covariance matrix is supplied for that group, the name of the pertinent covariance matrix file must follow the value of the group-specific measurement objective function target. Figure 9.4 shows an example.

```
* observation groups
obs_gp_1 45.00
obs_gp_2 67.00 covmat.dat
obs_gp_3 67.00
regul1
regul2
regul3
```

Figure 9.4 The "observation groups" section of a PEST control file, showing the use of group-specific measurement objective function targets.

9.6 A Final Word

Tikhonov regularisation, supported by singular value decomposition and LSQR as numerical solution devices, forms the backbone of highly parameterized inversion as applied to two- and three-dimensional spatial models such as groundwater and reservoir models. The same methodologies have been successfully applied in other contexts as well. These include geophysical data interpretation and calibration of land use and surface water models where regularisation is used to support regionalisation of model parameters.

As was stated in the preface of this manual, PEST is supported by a suite of groundwater and surface water utilities which expedite its use in these contexts. These are downloadable from the PEST web pages. Many of the groundwater utilities focus on construction of calibration datasets for highly parameterized inversion where families of so-called "pilot points" form the spatial parameterisation device of choice. See the PLPROC utility in particular.

Once calibration has been achieved, attention can be turned to calibration-constrained uncertainty analysis. PEST utilities discussed in part II of this manual, as well as pertinent

members of the Groundwater and Surface Water Utilities, can provide assistance in this phase of the modelling process.

10. SVD-Assist

10.1 Concepts

10.1.1 Super Parameters

Through use of PEST's "SVD-assist" methodology, the run-time burden of highly-parameterized inversion can be reduced considerably, this making some inversion problems numerically tractable which would otherwise be intractable. It achieves this through defining a set of so-called "super parameters", and estimating these instead of the native model parameters. These super parameters are defined through singular value decomposition of a weighted Jacobian matrix that includes all model parameters.

Mathematically, super parameters are the scalar projections of the original parameter field onto a set of orthogonal axes which span the calibration solution space. A solution to the inverse problem is obtained by estimating values for these scalar projections and then multiplying each of them by a unit vector oriented along the corresponding axis; these vectors are then added to form the estimated parameter set. Ideally, the number of super parameters required to achieve a good fit between field measurements and corresponding model outputs is equal to the dimensions of the calibration solution space. This can be considerably smaller than the number of parameters actually employed by the model. However they are optimally defined with respect to the information content of the calibration dataset. Hence they constitute optimal receptacles for the information content of that data. Furthermore, their number is the minimum required to hold that information. See section 6.2.6 of Doherty (2015) for full details.

Once a set of super parameters has been defined, PEST needs to undertake only as many model runs per iteration as the number of super parameters. This is because PEST calculates finite-difference derivatives with respect to these super parameters instead of calculating derivatives with respect to native model parameters. This is the reason why the use of SVD-assist promulgates such a large reduction in the numerical burden of highly parameterized inversion. Meanwhile, behind the scenes, PEST translates the values of super parameters to the values of native parameters. All that the user needs to know about super parameters is that these are the labour-saving device that PEST employs to make such gains in model run efficiency possible. Their actual values are irrelevant.

The use of SVD-assist as a solution mechanism for the inverse problem does not preclude the use of supporting Tikhonov regularisation. In fact, for reasons stated in the previous chapter and in Doherty (2015), the use of Tikhonov regularisation should always be encouraged where a problem is ill-posed, for it provides the mechanism for achieving a solution to that problem which is as much in harmony with expert knowledge as possible. This can then establish the minimum error variance credentials of that solution; the solution can then provide the optimal platform for post-calibration uncertainty analysis. Fortunately, use of SVD-assist does not require that regularisation relationships be expressed in terms of super parameters. Rather, these can be expressed in terms of native parameters in exactly the same way as they would have been expressed if solution of the inverse problem was sought without the assistance of SVD-assist. PEST does the translation. (Note that SVD-assisted inversion can take place without the use of Tikhonov regularisation if you wish. It is not an essential component of its implementation.)

10.1.2 Implementation Overview

In the discussion that follows it is assumed, for convenience, that PEST is being used to conduct regularised inversion. The base-parameter PEST control thus has a PESTMODE setting of "estimation" or "regularisation". However there is no reason why SVD-assist cannot be used as a solution procedure for PEST's predictive analysis and Pareto operations, though this will less often be the case. Hence PESTMODE in the base parameter PEST control file can be set to any of "estimation", "regularisation", "prediction" or "pareto".

An inverse problem does not have to be formulated with SVD-assist in mind. SVD-assist is only a solution strategy. The choice to use it is a matter of numerical convenience. Hence the suggestions for attainment of numerically stable regularised inversion in pursuit of a minimum error variance solution to the inverse problem of model calibration that were presented in previous chapters should be followed. In particular:

- Include as many parameters in the inversion process as are required to reflect the heterogeneity of the simulated system. This endows the inversion process with the flexibility it needs to let heterogeneity arise in places, and in ways, that respect the information content of the calibration dataset. Additionally, it endows the post-calibration uncertainty analysis process with an ability to explore the effects of parameter nonuniqueness on the uncertainties of management-critical predictions.
- Add Tikhonov regularisation to the inversion process, thereby providing fall-back positions for all parameters, and the means through which departures from those fall-back positions should be realized if warranted by the data. Set the target measurement objective function to a suitable value. This may be very low for initial PEST runs, or at a level which suppresses the arising of spurious heterogeneity induced by over-fitting in later PEST runs.
- Include a singular value decomposition or LSQR section in the PEST control file. Use of either of these solution devices guarantees numerical stability of the inversion process.
- Set RLAMBDA1 to 10 and RLAMFAC to -3. A fast-moving Marquardt lambda provides a last line of defence against numerical instability at the same time as it accommodates model linearity.
- Where model numerical performance is questionable, some of the other solution aids such as split slope analysis and model run failure forgiveness discussed in chapter 4 of this manual should also be considered.
- Ensure correct weighting relativity between different observation groups. Make sure that each group contributes about the same to the initial measurement objective function as any other group. (The PWTADJ1 utility described in part II of this manual can help in this regard.)

Once a PEST input dataset has been prepared, set the NOPTMAX variable in the "control data" section of the PEST control file to -2. Then run PEST. PEST will then calculate a Jacobian matrix based on native model parameters. This will be stored in file *case.jco*, where *case* is the filename base of the PEST control file. Immediately after it has stored this file, PEST will cease execution.

Next, decide on the number of super parameters that you would like to use (see below). Then run the SVDAPREP utility to build a new PEST control and ancillary files that implement SVD-assisted inversion. For convenience, let us refer to this file as *case-svda.pst*. You don't even need to look at this file if you don't want to, for its details are unimportant.

Nevertheless, for those who are interested, its specifications are outlined later in this chapter. Then run PEST on the basis of this new PEST control file by typing the command.

```
pest case-svda
```

Any version of PEST can be used, including Parallel PEST and BEOPEST.

The first iteration of an SVD-assisted PEST run happens very quickly, for PEST calculates super parameter derivatives from base parameter derivatives stored in *case.jco*. For all subsequent iterations, however, it calculates the derivatives of super parameters using the standard finite-difference procedures that were described earlier in this manual. As PEST undertakes an SVD-assisted model run it writes the same information to the screen as that which it would write for a normal PEST run. In particular, it records the measurement and regularisation objective functions, and contributions to the objective function by different observation groups, so that you can monitor PEST's performance as the inversion process advances. The same information is recorded in its run record file. Meanwhile other files such as the interim residuals file (this being named *case-svda.rei*) are also recorded so that PEST's progress in reducing specific residuals can be monitored if desired.

If conducting SVD-assisted inversion, PEST can be stopped and restarted using the restart switches described in section 5 of this manual, just as it can when conducting normal inversion.

At any stage of the inversion process best parameter values are stored in the parameter value file *case-svda.par*. However because it records the optimised values of super parameters, this file is of little use to the user; note that super parameters are named "par1", "par2", etc. The optimised values of base parameters are stored in a file named *case.bpa* (where "bpa" stands for "best parameter values"). Note that the filename base of this file is the same as that of the base PEST control file and not the super-parameter PEST control file.

When conducting SVD-assisted inversion the same run termination criteria are employed as those that are employed in a normal PEST run. As has been discussed, however, there is no need to wait for PEST to declare the inversion process as "over" if you feel that it has done enough. If this is the case, then you can stop PEST execution yourself in the usual way.

Even if PEST terminates execution of its own volition, it does not undertake a final model run using optimised parameters when implementing SVD-assisted inversion. This is because the relationship between super parameters and base parameters may change as the inversion process progresses; hence if best parameters were achieved on an early iteration the wherewithal to convert current super parameter values to base parameter values has been lost. Fortunately it is an easy matter to undertake a model run manually based on best parameters achieved through SVD-assisted inversion. Simply use the PARREP utility (see part II of this manual) and a command such as the following to create a new base PEST control file (in this case denoted as *case-soln.pst*) in which initial values are optimised parameters.

```
parrep case.pst case.bpa case-soln.pst
```

Then set NOPTMAX to zero in file *case-soln.pst* and run PEST. PEST will then run the model once on the basis of optimised parameters before ceasing execution. Alternatively, you may wish to use the SUBREG1 utility to remove regularisation from *case-soln.pst* and then set NOPTMAX to -2 in this file to compute a new Jacobian matrix. The use of the PWTADJ2 utility may also be considered as a means of adjusting weights in this new PEST control file so that they reflect the stochastic character of model-to-measurement misfit. You are then ready to undertake post-calibration linear parameter and predictive uncertainty analysis using

utilities such as GENLINPRED described in part II of this manual.

10.1.3 How Many Super Parameters?

How many super parameters should you employ? Use as many as computing resources permit.

The SUPCALC utility described in part II of this manual attempts to calculate the optimal dimensions of the solution space for a particular inverse problem. It has the information that it needs to do this once a Jacobian matrix has been calculated using base parameters. However there is no need to limit the number of super parameters that you include in an SVD-assist PEST control file to the SUPCALC-calculated dimensions of the solution space. As is discussed below, definition of super parameters is based on a linearity assumption that is probably violated to a greater or lesser degree by most models. The dimensionality and composition of solution and null spaces may change as parameter values change. The more super parameters that are estimated, the less this matters. In the end, the only consideration on the number of super parameters to use is the computing resources at your disposal. In fact, if you have enough computing resources to conduct normal inversion, then there is no need to use SVD-assist as a numerical device for solution of the inverse problem at all.

By definition, if the number of super parameters exceeds the dimensions of the solution space, then the super parameter estimation problem is ill-posed. Normally this does not matter. Tikhonov regularisation, if employed, guides the inverse problem towards a minimum error variance solution. Furthermore, if singular value decomposition or LSQR is designated as the solution mechanism in the base parameter PEST control file, then the SVDAPREP utility (see below) designates this as the solution mechanism in the super parameter PEST control file. Numerical stability of the SVD-assisted inversion process is therefore guaranteed, regardless of the number of super parameters which are featured in this problem.

Sometimes available computing resources may be insufficient for use of even the minimum number of super parameters that SUPCALC suggests. Then use whatever number of super parameters that you can. As is explained by Doherty (2015), the parameter simplification implied in definition of super parameters is optimal in the sense that it provides maximum receptivity to the information content of the calibration dataset using the fewest number of "parameter receptacles" for that information. In the end, environmental modelling, and environmental model calibration, is a matter of compromise. Super parameters comprise a very good compromise.

10.1.4 Strengths and Weaknesses

The great strength of the SVD-assist methodology is the huge savings in model runs that its use accrues. The smaller are the dimensions of the calibration solution space with respect to the number of parameters featured in the inversion process, the greater are these potential savings.

The great weakness of the SVD-assist methodology has already been mentioned. This is the fact that super parameter definition is based on a linearity assumption that may be violated by the model. As was stated above, some defence against nonlinearity can be gained through use of a larger number of super parameters than is actually required to span the solution space.

An alternative defensive strategy is to repeat the super parameter definition process after a number of iterations of the SVD-assisted inversion process has occurred. Suppose, for example, that PEST made little progress in reducing the measurement objective function on

the third iteration of this process. (Check first that the reason for this is not imposition of relative or factor limits on super parameter movements, or many base parameters hitting their bounds.) You may then decide to halt PEST execution and, using the PARREP utility, create a new base parameter PEST control file featuring best parameters obtained so far from the stalled SVD-assisted inversion process. With NOPTMAX set to -2 in this new file, run PEST to compute a new base parameter Jacobian matrix. Then use SVDAPREP to construct a new super-parameter PEST control file. Run PEST using that file to resume the SVD-assisted inversion process.

Further steps can be taken to ameliorate the effects of model nonlinearity. As always, initial parameter values in the base PEST control file should be preferred values from an expert knowledge point of view. A solution to the Tikhonov-regularised inverse problem then seeks a parameter set which, on the one hand, fits the calibration dataset while, on the other hand, deviates minimally from this preferred parameter set.

Sometimes, especially in the groundwater and reservoir modelling contexts, it may be difficult for a user to decide on preferred values for all parameters cited in a PEST control file. However it may be possible to give SVD-assisted inversion a head start in other ways. Suppose, for example, that parameters of a multilayer model are being estimated and that each layer is endowed with many parameters (possibly hundreds of parameters). An initial parameter estimation run may be undertaken wherein all parameters in each layer are tied to a single parameter that thereby represents the entire layer. This reduces the dimensionality of the inverse problem to the number of layers in the model. If the initial values of parameters are all the same in each layer, this strategy embodies an assumption of layer property uniformity. Once the limited number of layer-specific parameters defined in this manner have been estimated, parameters can be untied from each other, Tikhonov regularisation can then be introduced (possibly encapsulating layer-uniformity parameter constraints), and a Jacobian matrix can be computed for all model parameters by setting NOPTMAX to -2 and running PEST. SVD-assisted inversion can then proceed in the manner described above.

The enforcement of bounds on base parameters is another area where compromises must be made in implementing SVD-assisted inversion. Bounds that are set in a base parameter PEST control file are respected when PEST estimates super parameters. If a super parameter upgrade vector is such that a base parameter will hit its bound, then PEST shortens the upgrade vector so that this does not occur. (This can limit the extent to which the measurement objective function can be improved on that iteration.) PEST then re-defines super parameters with the base parameter removed from the inversion process and permanently attached to its bound. This has two disadvantages. The first is that super parameter re-definition can be numerically intensive (and therefore slow) where base parameter numbers are high. The second disadvantage is that a thus-bounded parameter is given no opportunity for further adjustment. Sadly, however, there is no alternative but to adopt this strategy if super parameters themselves are to remain open to adjustment, and with them the base parameters that they collectively encapsulate.

It follows that bounds in a base parameter PEST control file should be set wide if this is possible.

10.2 SVDAPREP

10.2.1 Preparing a PEST Input Dataset

As was stated above, the first step in SVD-assisted inversion is the same as the first step in

any inversion, namely the construction of a PEST input dataset. This, of course, will include a control file, possibly with Tikhonov regularisation introduced to that file. This file can include fixed and tied parameters if desired. Individual parameters can be log-transformed or not transformed at all. Once prepared, the PEST input dataset should be checked using the PESTCHEK utility. This PEST control file is referred to as the "base parameter PEST control file".

NOPTMAX should then be set to -2 and PEST run in order to compute a Jacobian matrix.

Once the Jacobian matrix has been computed the SVDAPREP utility should then be run in order to construct a PEST dataset based on super parameters. (Note that if you change your mind and decide not to use SVD-assist as an inverse problem solution methodology after all, this does not mean that you have just wasted the model runs required to compute the Jacobian matrix. Set NOPTMAX to a higher number in the PEST control file and start PEST with the "/i" switch. As described earlier in this manual, PEST will then prompt for the name of a Jacobian matrix to use for its first iteration.)

While the base parameter PEST control file can be just like any other PEST control file, SVDAPREP demands that one specification be respected. (If this is not the case it will cease execution with an appropriate error message.) The command to run the model must end in "*.bat*". In WINDOWS this signifies a batch file. This naming convention does not signify a script file in UNIX; nevertheless it is required by SVDAPREP.

The "model" which PEST runs must, indeed, be a batch or script file. This is because SVDAPREP must make a copy of this file and add some lines to the top of it. These lines issue commands to run the PARCALC and PICALC utilities which transform super parameters to base parameters, and to evaluate prior information equations expressed as relationships between base parameters. It is important that these utility programs reside in a directory (i.e. a folder) that is cited in the PATH environment variable so that the operating system knows where to find them when the command to run them is issued. Alternatively a copy of the PARCALC and PICALC executable programs should be placed in the PEST working directory (and in the working directory of each of its slaves if Parallel PEST or BEOPEST is used for solution of the inverse problem).

10.2.2 Running SVDAPREP

SVDAPREP is run by typing its name at the screen prompt. On most occasions that it is run, you can respond to many of its prompts simply by pressing the <Enter> key, as this signifies acceptance of a respective default.

On commencement of execution, SVDAPREP asks you for the name of the base parameter PEST control file.

```
Enter name of existing PEST control file:
```

(If you omit the "*.pst*" extension from the name of the PEST control file, SVDAPREP provides this extension itself.) SVDAPREP then reads this file. It also searches for a file named *case.jco* (where *case* is the filename base of the base parameter PEST control file) which contains the base parameter Jacobian matrix calculated by running PEST in the manner discussed above. It next asks

```
Use pre-defined super-parameter file? [y/n]  (<Enter> if "no"):
```

If your response to this prompt is "y", SVDAPREP will then prompt for the name of a super parameter file. It will set the SVDA_EXTSUPER control variable to 1 in the super parameter

PEST control file that it writes, and will record the name of the super parameter file in this new PEST control file. These and other features of the super parameter PEST control file are described later this section.

If you do not request that super parameters are externally defined (which is normally the case), SVDAPREP's next prompt is

```
For computation of super parameters:-
    if SVD on Q^(1/2)X                - enter 1
    if SVD on XtQX                    - enter 2
    if LSQR without orthogonalisation - enter 3
    if LSQR with orthogonalisation    - enter 4
```

In these prompts **X** is the Jacobian matrix pertaining to base parameters (normally represented as **J** in this manual), with any regularisation removed from this matrix. **Q** is the observation weight matrix. If implemented, LSQR is applied to $\mathbf{Q}^{1/2}\mathbf{J}$.

Option 1 is best unless there are more than about 2500 base parameters. In this case choose option 3 or option 4 as singular value decomposition becomes very slow when parameter numbers start to rise. Note that SVDAPREP does not calculate super parameters itself. Rather it informs PEST how it must calculate them through the setting of appropriate control variables in the super parameter PEST control file (see below). PEST calculates super parameters at the beginning of the SVD-assisted inversion process, and whenever a base parameter hits its bound thereafter.

Next SVDAPREP asks

```
Enter number of super parameters to estimate:
```

Enter an appropriate number. See the discussion above.

SVDAPREP next prompts

```
Enter name of new super pest control file:
```

Provide a name for the new PEST control file. (If you omit the "*.pst*" extension from the name of the PEST control file, SVDAPREP will provide this extension itself.) This new PEST control file will be used for SVD-assisted parameter estimation. Next SVDAPREP prompts

```
Enter offset for super parameters  (<Enter> if 10):
```

In the SVD-assisted inversion process, super parameters are provided with a starting value of zero (signifying zero perturbation of initial base parameters). However zero-valued parameters can create problems for PEST, especially in the enforcement of parameter change limits. Hence it is best to supply an offset for such parameters, to keep their values away from zero. A value of 10 is suitable on most occasions of SVD-assisted parameter estimation; simply press <Enter> for SVDAPREP to accept this.

Super parameters are designated as "relative limited" by SVDAPREP. As described in chapter 4 of this manual, the actual value of this relative limit must be supplied as the PEST variable RELPARMAX. It is important to set this variable much lower than would normally be the case, but not so low that parameter upgrades are unusably small. On most occasions a value of 0.1 is adequate, though you should be prepared to alter this upwards if PEST convergence is too slow, or downwards if parameter oscillation occurs, or parameters hit their bounds too quickly. SVDAPREP asks the user for a suitable value for RELPARMAX to record in the control file that it is about to write:

```
Enter value for RELPARMAX (<Enter> if 0.1):
```

Simply press <Enter> to accept the default value of 0.1

Note that the above defaults of 10 for super parameter OFFSETs and 0.1 for RELPARMAX are somewhat conservative. However they work well on most occasions. Often faster SVD-assisted inversion can occur (with a small risk of numerical malperformance) if the OFFSET is set to 20 and RELPARMAX is set to 0.2. This is often worth a try.

SVDAPREP's next prompt is

```
Write multiple BPA, JCO, REI, none [b/j/r/n] files (<Enter> if "n")?
```

An appropriate response to this prompt can instruct PEST to write iteration-specific base-parameter value files, Jacobian matrix files and/or intermediate residual files when it undertakes SVD-assisted inversion.

Iteration-specific super parameter Jacobian matrix files are rarely needed; they can sometimes be of use in exploring the integrity (or otherwise) of finite-difference derivatives. Iteration specific interim residuals files are also little used. Iteration-specific base parameter value files can be useful where the target measurement objective function is set low and a user feels that he/she would like to select as his/her optimised parameter values those which were computed before parameters showed obvious signs of over-fitting.

Next SVDAPREP asks

```
Parameter scale adjustment [SVDA_SCALADJ] setting (<Enter> if 2):
```

The SVDA_SCALADJ parameter is discussed in detail below. A setting of 2 is best for most occasions.

The next prompt is

```
Make new model batch file silent or verbose?  [s/v] (<Enter> if "s"):
```

As is discussed below, SVDAPREP writes a new model batch file. Screen output from executable programs added to this file for the purposes of base parameter calculation and possibly prior information calculation can be suppressed if desired (thus avoiding mixing of output from these programs with PEST's screen output if non-parallel versions of PEST are used); respond to the above prompt by pressing the "s" or "v" key as appropriate.

SVDAPREP's final prompt is

```
Automatic calc. of 1st itn. super param. derivs?  [y/n] (<Enter> if "y"):
```

A response of "y" or simply <Enter> to this prompts instructs SVDAPREP to supply a value of 1 for the SVD-assist control variable SVDA_EXTSUPER. The derivatives of super parameters are therefore calculated from those of base parameters during the first iteration of the SVD-assisted parameter estimation process. This saves PEST from having to calculate these derivatives using finite differences, and hence saves model runs.

10.2.3 What SVDAPREP Does

Once it has received responses to all of these prompts, SVDAPREP undertakes the following tasks.

1. If prior information is present in the original PEST control file, SVDAPREP writes a file named *picalc.tpl*. This is the template file for the input file *picalc.in* of program PICALC, run as part of the new model for the calculation of prior information using

base parameters. Outcomes of these calculations are written by PICALC to a file named *picalc.out*.

2. If prior information is present in the base parameter PEST control file, SVDAPREP writes an instruction file named *picalc.ins* to read the outcomes of PICALC's prior information calculations. "Observations" cited in this file correspond to the names of prior information equations cited in the original base PEST control file.
3. SVDAPREP writes a new PEST control file for use in the SVD-assisted parameter estimation process. This contains the same observations (with the same weights) as those contained in the original base PEST control file. However it contains new parameters, these being named "par1" to "par*m*" where *m* is the number of super parameters requested by the user. These are provided with an initial value equal to the user-supplied OFFSET, and an OFFSET equal to the negative of this value, thus endowing these parameters with an effective initial value of zero. Derivatives with respect to these parameters are calculated using an absolute increment of 0.015 (you can alter this if you wish). Super parameters are relative-limited, the value of this limit having been supplied through the pertinent SVDAPREP prompt as described above. The new PEST control file is instructed to run a new model named *svdabatch.bat*; this is altered from the original model batch file cited in the base PEST control file in the manner discussed below. Also, in generating the new PEST control file, all prior information is re-assigned as observations, the model-counterpart to these observations being calculated by PICALC. Observation group names are preserved in this re-assignment process so that any regularisation implemented in the original PEST control file can still be implemented in the new SVD-assist PEST control file. This latter file is also provided with an "svd assist" section. Finally, the new PEST control file cites only a single model template file, this being named *parcalc.tpl*. This is used to construct a model input file named *parcalc.in* for the PARCALC utility, run as part of the model to implement translation from super parameters to base parameters. The actual template file itself is written by PEST (for this file needs to be altered as new combinations of base parameters are used to calculate super parameters as base parameters hit their bounds). Also PARCALC is responsible for writing normal model input files on the basis of base parameter values, a role previously assumed by PEST; the SVD-assisted PEST control file cannot do this, as it does not know the values of base parameters – only the values of super parameters.
4. SVDAPREP writes a new model batch file, the details of which will now be described.

10.2.4 The New Model Batch File

The model batch file, run through the new SVD-assist PEST control file, is altered from the original model batch file cited by the base PEST control file in the following ways.

1. Model input files written by PARCALC on the basis of previously-supplied template files are deleted prior to running PARCALC; this prevents old files from being re-read as new ones in the event that PARCALC does not run.
2. The command to run PARCALC is included in the modified batch file.
3. If prior information is present in the original PEST control file, the command to run PICALC is included in the modified batch file.

4. If the user requested that screen output be suppressed, the "> nul" string ("> /dev/null" in the UNIX environment) is added to all new commands; the "@echo off" string is also added to the top of this file. (Note that these measures do not suppress screen output from existing components of the original model batch file which are transferred directly to the new model batch file *svdabatch.bat*.)

As was mentioned above, in the original base PEST control file the command to run the model must be the name of a batch file – not the name of an executable program; SVDAPREP needs to modify this batch file to produce *svdabatch.bat.* Consequently, if the suffix "*.bat*" does not comprise the last four letters of the model command recorded in the base parameter PEST control file, SVDAPREP will cease execution with an appropriate error message.

10.3 The PARCALC and PICALC Utilities

10.3.1 PARCALC

You never need to run the PARCALC utility program. This is part of the model run by PEST when implementing SVD-assisted parameter estimation. Nevertheless its functionality will be described.

PARCALC reads the current values of super parameters being estimated by PEST. It also reads base parameter singular vectors (i.e. eigenvectors) as calculated by PEST on the basis of the JCO file cited in the "svd assist" section of the super PEST control file; recall that super parameter definition is based on these singular vectors (unless super parameters are user-supplied). Using this information, PARCALC calculates current base parameter values. These are then provided to the model through the template files what were originally used by the base PEST control file.

In calculating current base parameter values, PARCALC does the following.

1. It respects upper and lower base parameter bounds as provided in its input file *parcalc.in*, these having been extracted from the base parameter PEST control file.
2. The values of fixed base parameters are unaltered. (These parameters are not used in the calculation of super parameters.)
3. Parameter value ratios for tied parameters are maintained. (Tied parameters are also not used in the calculation of super parameters.)

PARCALC is run by typing the command

```
parcalc
```

at the screen prompt. It reads an input file named *parcalc.in.* This file is actually written by PEST on the basis of the template file *parcalc.tpl*, which is also written by PEST. Use of the template file mechanism for the provision of PARCALC input data allows parallel PEST and BEOPEST to transfer PARCALC input data to multiple computers used for parallelisation of model runs.

10.3.2 PICALC

PICALC requires no user input. It reads an input file named *picalc.in* and writes an output file named *picalc.out*. The first of these files is written by PARCALC on the basis of the SVDAPREP-prepared template file *picalc.tpl*. This contains current values of base

parameters, as well as parameter coefficients used in prior information equations. PICALC evaluates these equations on the basis of current base parameter values and records the outcomes of these calculations to a file named *picalc.out*.

10.4 The Super Parameter PEST Control File

10.4.1 General

In many ways the PEST control file used for SVD-assisted inversion is like any other PEST control file. It is subdivided into the usual sections, but has one extra section, this being the "svd assist" section. It does not possess a "prior information" section, however. As is discussed above, prior information equations in the base parameter PEST control file become observations in the super parameter PEST control file. Modelled values of these observations are calculated by the PICALC utility. However as their group names are retained, they can still embody Tikhonov regularisation (and are treated accordingly by PEST) if these names begin with "regul".

Super parameters are given the names "par1", "par2",…, "par*m*" where *m* is the number of super parameters that PEST estimates in the SVD-assisted inversion process. (Actually PEST's SVD-assist functionality does not require that they be given these names; these are the names that they are given by SVDAPREP.)

10.4.2 An Important Note on Super Parameter Definition

Unless it is supplied with externally-calculated super parameters (see below), PEST calculates super parameters itself from information provided in the base parameter control file on the one hand, and the Jacobian matrix file calculated on the basis of the base parameter control file, on the other hand. It removes rows from the base parameter Jacobian matrix file that pertain to regularisation; thus definition of super parameters takes account only of the information content of the calibration dataset, and not of information contained in any expert knowledge that may be expressed through regularisation constraints.

If the base PEST control file instructs PEST to run in "predictive analysis" mode, then the row of the Jacobian matrix that contains sensitivities to the user-defined prediction is removed from this matrix prior to calculation of super parameters.

Observation weights as represented in the base parameter PEST control file are taken into account in defining super parameters. Numerically, super parameters are calculated through singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ (or equivalently $\mathbf{J}^t\mathbf{Q}\mathbf{J}$) where $\mathbf{J}$ is the base parameter Jacobian matrix (with regularisation removed) and $\mathbf{Q}$ is the base parameter weight matrix. (Using LSQR instead of singular value decomposition achieves the same thing.)

When SVDAPREP constructs a super parameter PEST control file, observation values and weights are transferred from the base parameter PEST control file to the super parameter PEST control file. However there is nothing to stop you from altering observation values and weights recorded in the super parameter control file once this file has been built by SVDAPREP. Actually, if you wish, you can define a completely new set of observations altogether, and give them whatever weights you wish. Super parameter definition still takes place on the basis of observations and weights contained in the original base parameter control file despite the fact that the values of super parameters (and therefore of base parameters) are being estimated using a new set of observations and weights.

There may be some situations where the ability to undertake super parameter definition on

the basis of one set of observations and weights, and parameter estimation on the basis of another set of observations and weights is important. A simple but important instance where this may prove useful is where a covariance matrix is supplied for super parameter estimation based on the observation correlation structure induced by using super parameters. As discussed by Doherty (2015), all forms of regularisation (whether manual or mathematical) promulgate correlated "measurement noise". If it is desired that account be taken of the statistical structure of this noise in the parameter estimation process, SVDAPREP can be used for generation of the super parameter PEST control file in the usual fashion. Then an appropriate covariance matrix can be supplied for the observations contained in that file. However if individual weights, rather than a covariance matrix, are used in the base PEST control file, super parameter definition takes place on the basis of individual weights, while estimation of super parameters (and hence indirectly of base parameters) takes place using a covariance matrix which best characterizes "measurement noise" in these circumstances. In more complex modelling contexts, the number, names and types of observations can be different in the super parameter PEST control file from those contained in the base parameter PEST control file.

It is important to note that, as presently programmed, PEST does not allow a covariance matrix to be supplied with non-regularisation observations in the base parameter PEST control file. However this restriction does not apply to the super parameter PEST control file.

10.4.3 The "SVD Assist" Section

Specifications of the "svd assist" section of the PEST control file are provided in figure 10.1. If present, this section must be placed just before the "parameter groups" section of the PEST control file following the optional "singular value decomposition" or "lsqr" sections. PEST knows that it is implementing SVD-assisted inversion through the presence of this section in the PEST control file.

Variables appearing on the fourth line of the "svd assist" section are all optional (though their appearance is recommended). The presence of any one of these variables requires that those preceding it on this line also be present so that PEST knows which variable it is.

```
* svd assist
BASEPESTFILE
BASEJACFILE
SVDA_MULBPA SVDA_SCALADJ SVDA_EXTSUPER SVDA_SUPDERCALC SVDA_PAR_EXCL
```

Figure 10.1 Specifications of the "svd assist" section of the PEST control file.

BASEPESTFILE

BASEPESTFILE is the name of the PEST control file on which the inversion process is based; this is the PEST control file in which all base parameters are cited. The base PEST control file is a normal PEST control file. It may instruct PEST to run in "estimation", "predictive analysis", "regularisation" or "pareto" modes. It may or may not contain prior information. It may have been constructed "by hand", it may have been written with the help of programs from the PEST Groundwater or Surface Water Utilities suite, or it may have been constructed using third party software. It should not, however, contain a "svd assist" section.

BASEJACFILE

This is the name of a Jacobian matrix file (i.e. a JCO file). This will normally have been

produced by running PEST on the basis of the nominated base parameter PEST control file, with NOPTMAX in that file set to -2. Note, however, that if PEST uses externally-defined super parameters, then BASEJACFILE must not be a JCO file at all; in this case it must be an appropriately formatted file containing externally-calculated super parameters. This is further discussed below.

SVDA_MULBPA

SVDA_MULBPA can be set to either 0 or 1. If it is set to 1, then not just one, but a series of "bpa files" are recorded during the course of the SVD-assisted parameter estimation process. Recall that a "bpa file" is a parameter value file pertaining to base parameters. At any stage of the inversion process it contains best base parameter estimates calculated up until that point; at the end of the inversion process it contains best base parameters achieved throughout the entire process. As was explained above, the filename base of the bpa file is that of the base parameter PEST control file, not the super parameter control file through which parameter estimation actually takes place.

With SVDA_MULBPA set to 1, PEST records two bpa files on every occasion that base parameter values are improved. The first of these files is the normal bpa file. The second has the same name, but with an extension of "*.N*" appended, with "*N*" indicating the iteration number in which the base parameters recorded therein were achieved. Thus if, for example, the filename base of a base PEST parameter control file is *case*, then at the conclusion of the parameter estimation process bpa files named *case.bpa*, *case.bpa.1*, *case.bpa.2* etc. will be present within the PEST working directory. The first of these contains the overall best base parameters achieved through the entire inversion process. The others contain base parameter values achieved during the indicated iterations. Note that not all iterations of the inversion process will be represented in this sequence – only those iterations will be represented where base parameters were improved.

SVDA_SCALADJ

Problems can arise when log-transformed and untransformed parameters are simultaneously estimated using SVD-assist. These problems are an outcome of the fact that log-transformed parameters can have vastly different sensitivities from those of untransformed parameters. In fact the latter can have vastly different sensitivities from those of other untransformed parameters. Special considerations associated with the handling of very different sensitivities arise when implementing SVD-assisted parameter estimation.

Ideally, when formulating an SVD-assisted parameter estimation problem, parameters should be normalized by their standard deviations prior to super parameter definition. (Actually, as described by Doherty (2015), ideally they should be subjected to Kahunen-Loève transformation; if parameters show prior correlation then normalization by standard deviations is only a rough approximation to Kahunen-Loève transformation.) There are a number of reasons for this. One of them is that solution to the regularised inverse problem may then approach the solution of minimum error variance. (if initial parameter values are chosen wisely). Another important consideration is that normalisation by standard deviation tends to equalize sensitivities across different parameter types. If, after such normalisation, a particular base parameter, or group of base parameters, is found to be relatively insensitive, such parameters will be removed from consideration in the normal course of formulating the SVD-assisted parameter estimation process as they will not feature strongly in super parameters. On the other hand, base parameters which are particularly sensitive after

normalisation will feature strongly in super parameters, and hence will be adjusted during the parameter estimation process.

If non-log-transformed base parameters are not normalised, it is possible that some of them may become "artificially hypersensitive". For example in a groundwater model, recharge (which may be left untransformed to increase the linearity of the inversion process) is often far more sensitive than other parameters (often by many orders of magnitude) simply because low numbers are used to represent it in many modelling contexts. Thus recharge base parameters will feature strongly in super parameters. Furthermore, in many cases, the top few super parameters may then be hundreds of times more sensitive than the next super parameters, this creating numerical problems for PEST as it tries to adjust both the recharge-dominated super parameters, together with super parameters of far lower sensitivity which contain linear combinations of other base parameters whose adjustment may be far more effective in lowering the objective function than recharge parameters. If recharge parameters were normalized with respect to their natural variability this problem would not occur, for a very low number expressing what may be a comparatively high recharge value would then become a comparatively high number. Because variations in this high number then have smaller effects on model outputs than variations in the corresponding unnormalised very small number, the extreme sensitivities associated with the latter are removed.

A similar affect could be achieved if recharge were log-transformed. Mathematically, derivatives of log-transformed parameters are automatically normalized by their values – this achieving something approaching normalization by natural variability. However, in the case of recharge, while log transformation may remove artificial hypersensitivity, the parameter estimation process may suffer because of the nonlinearities introduced to it through log transformation of recharge.

Another problem that arises with "artificial hypersensitivity" of base parameters is that it is far too easy for them to encounter their bounds in the course of the parameter estimation process. Recall that it is an unfortunate necessity of employing SVD-assist as an inversion mechanism that once a base parameter hits its bound, it must stay there forever. Premature bounds collisions can be mitigated by employing a relatively low RELPARMAX setting. However even this measure cannot forestall wholesale bounds collision of hypersensitive parameters, often on the first iteration of the parameter estimation process. The reason for this is that super parameters are actually the coefficients by which normalised singular vectors of $\mathbf{Q}^{1/2}\mathbf{J}$ are multiplied in determining parameter sets which best reduce the measurement objective function. The component of a hypersensitive base parameter in one of these normalised singular vectors can be such that, even when the corresponding super parameter is increased only incrementally for the purpose of derivatives calculation, the pertinent base parameter can hit its bounds; this happens solely as a consequence of the fact that the singular vector has a magnitude of 1 and that hypersensitive base parameters (which will dominate the top few singular vectors) have a "natural magnitude" which may be orders of magnitude smaller than this. Obviously this is an outrageous situation which must be prevented at all costs. Fortunately, the problem is easily forestalled by estimating high multiples of hypersensitive base parameters - that is, by appropriate base parameter scaling.

Base parameter scaling can be easily undertaken by direct editing of a base PEST control file prior to running SVDAPREP. Say, for example, that all parameters of a certain type (for example all recharge parameters) are to be scaled so that they are a thousand times less sensitive, thus avoiding the deleterious effects of artificial base parameter hypersensitivity discussed above. This could be achieved in the following manner.

1. Provide all recharge parameters with a SCALE of 0.001 in the "parameter data" section of the base PEST control file.
2. Multiply all recharge parameter initial values, and their bounds, by a factor of 1000.
3. Multiply any absolute derivative increments, and relative increment absolute lower bounds, by a factor of 1000 in the "parameter groups" section of the PEST control file.
4. Alter any prior information equations containing these parameters such that they are correct when any recharge parameter involved in any equation is multiplied by 1000.

Thus PEST "sees" (and estimates) 1000 times each recharge parameter, while the model "sees" correct recharge parameter values. (Note that if an OFFSET is employed for a parameter this does not need to be altered because PEST multiplies by the SCALE before it adds the OFFSET. Note also that if these changes are made to a base parameter control file for which a corresponding JCO file already exists, the JCO2JCO utility described in part II of this manual can be used to create a new JCO file corresponding to the edited PEST control file, thus obviating the need for re-calculation of base parameter sensitivities). As a result of these alterations to the base PEST control file, sensitivities with respect to these now-scaled recharge base parameters will be 1000 times smaller than those of their unscaled counterparts.

Fortunately, equivalent operations can be performed automatically as a practical means of ensuring good performance of SVD-assisted inversion by setting the SVDA_SCALADJ control variable appropriately.

Permissible values for SVDA_SCALADJ are -4, -3, -2, -1, 0, 1, 2, 3, and 4. (If omitted from the PEST control file, a default value of 0 is assumed; in this case no base parameter scaling is undertaken.) Base parameter scaling options activated by SVDA_SCALADJ are now discussed in detail. It is important to note that all scaling is undertaken internally by PEST. Estimated parameter values as recorded, for example, in the ***case**.bpa* parameter value file do not require "de-scaling" by the user.

(i) Non-Log-Transformed Parameter Scale Adjustment by Group

As stated above, problems caused by base parameter hypersensitivity do not tend to occur when implementing SVD-assisted parameter estimation where all base parameters are log-transformed. This is a result of the "self-normalizing" effect of logarithmic transformation. One option for untransformed parameters is thus to scale them such that their sensitivities are roughly equal to those of log-transformed parameters involved in the inversion process. If SVDA_SCALADJ is set to 1 PEST undertakes scaling of untransformed base parameters in the following fashion.

1. First it calculates the composite sensitivity of every adjustable base parameter; this is calculated as the magnitude of the column of the Jacobian matrix corresponding to each parameter, with each entry in that column multiplied by the squared weight associated with the corresponding observation.
2. Next it calculates the average composite sensitivity of all parameters in each parameter group; that is, it calculates a kind of parameter group composite sensitivity. It is assumed that all adjustable parameters within any parameter group are either log-transformed or untransformed (tied and fixed members of a particular parameter group are permitted).

3. For each untransformed parameter group PEST evaluates a scaling factor, applied to all parameters in the group, which equates the average composite sensitivity for that group to that of the log-transformed group of maximum average composite sensitivity.

Variations of this theme are as follows.

1. If SVDA_SCALADJ is set to -1, PEST will never *increase* the sensitivity of an untransformed parameter group to ensure composite sensitivity equalisation with the log-transformed parameter group of highest average composite sensitivity; it will only *decrease* the sensitivity of any untransformed parameter group.
2. If SVDA_SCALADJ is set to 2, PEST calculates the average composite sensitivity of *all* log-transformed parameters, irrespective of the parameter group to which they belong. Then, for each untransformed parameter group, all member parameters are scaled such that the average composite sensitivity of the parameter group is equal to that of all log-transformed parameters.
3. If SVDA_SCALADJ is set to -2, PEST will never *increase* the sensitivity of an untransformed parameter group to ensure composite sensitivity equalisation with all log-transformed parameters; it will only *decrease* the sensitivity of an untransformed parameter group.

PEST will terminate execution with an error message if SVDA_SCALADJ is set to -2, -1, 1 or 2 and any of the following conditions are met.

1. A base parameter group exists which contains both log-transformed and untransformed parameters.
2. No log-transformed parameters are cited within the base control file.
3. All log-transformed base parameters have zero sensitivity.

(ii) Individual Non-Log-Transformed Base Parameter Scale Adjustment

As stated above, ideally parameters should be scaled by their standard deviations - that is, by their innate variability – that is, by the uncertainty associated with those parameters in the absence of any calibration information, this uncertainty being based solely on specialist knowledge of the system. Such information cannot be supplied directly on a PEST control file. However parameter upper and lower bounds (which *are* supplied through a PEST control file) can be considered as surrogates for information of this type.

If SVDA_SCALADJ is set to 3, PEST assumes that the upper and lower bounds of all untransformed parameters are set approximately four standard deviations apart. Untransformed parameters are then scaled by standard deviations calculated accordingly. (There is no need for exactness here as the success of scaling in improving the performance of SVD-assisted inversion is not unduly sensitive to the exact scaling values. Four standard deviations is a convenient assumption as, for normally distributed parameters, it corresponds to a confidence interval of about 95 percent.)

If SVDA_SCALADJ is set to -3, PEST will not apply a scale factor that *increases* the sensitivity of a parameter; it will only apply a scale factor that *reduces* the sensitivity of an individual untransformed base parameter.

(ii) Pervasive Base Parameter Scale Adjustment

If SVDA_SCALADJ is set to 4, the strategy implemented when SVDA_SCALADJ is set to 3 is extended to accommodate all adjustable base parameters – even those that are log-transformed. As for untransformed parameters, it is assumed that the bounds on log-transformed parameters represent a separation of about 4 standard deviations, and that their probability distributions are log-normal. Once again, these assumptions are not critical by any means; they simply represent a convenient basis for scaling. If SVDA_SCALADJ is set to -4, no base parameter sensitivities will be increased in the process of base parameter scale adjustment.

Which Option to Use

Option 4 is the preferred option, both theoretically and in terms of optimisation of PEST performance when carrying out SVD-assisted parameter estimation. However it requires that the user pay more careful attention to the setting of parameter bounds than he/she may normally do. In fact on many occasion of PEST usage, scant attention is given to parameter bounds at all, these being set wide enough apart for PEST to "listen to what the data says" about the system being simulated by the model, and about the model itself, through the values that it assigns to parameters employed by the model, unencumbered by the necessity to respect bounds placed on these parameters. This, indeed, can be an important role played by the calibration process, for if PEST insists on assigning unacceptable values to certain model parameters, this may indicate conceptual flaws in the model. If all parameters are log-transformed, setting SVDA_SCALADJ to 4 with parameter bounds set wide apart should not cause problems (though it would be better to set it to zero under these conditions). However if some parameters are not log-transformed and the bounds on some of these parameters are set far apart, use of option 4 could lead to dreadful performance of the SVD-assisted parameter estimation process, as hypersensitivity of one or more types of untransformed parameter could result in rapid bounds collision for the reasons outlined above.

Notwithstanding superior performance of a properly-constructed SVD-assisted inversion process with SVDA_SCALADJ set to 4, the SVDAPREP utility described above provides a default setting of 2 for SVDA_SCALADJ. Experience has shown that SVD-assisted inversion when both log-transformed and untransformed parameters are simultaneously estimated is not as robust with this option as it is for options 3 or 4 with careful consideration given to the setting of base parameter bounds. However if base parameter bounds are set far apart in order to reduce their effect on the parameter estimation process (either deliberately for the reasons outlined above, through user-laziness, or even as a result of the unlikely, but not impossible, occurrence that a user did not read this section of the manual), then use of options 3 or 4 may have disastrous consequences for the SVD-assisted inversion process. Hence option 2 is the safest SVDAPREP option. The user should not forget, however, the benefits of setting SVDA_SCALADJ to 4 in conjunction with a careful bounds setting strategy.

SVDA_EXTSUPER

The SVDA_EXTSUPER control variable stipulates the manner in which super parameters are computed; it must be set to 0, 1, 2, -2 or 3.

If SVDA_EXTSUPER is set to 0 PEST computes super parameters on the basis of singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ where $\mathbf{J}$ is the base parameter Jacobian matrix read from the base parameter JCO file. However if SVDA_EXTSUPER is set to 3 PEST computes super parameters on the basis of $\mathbf{J}^t\mathbf{Q}\mathbf{J}$. While these two procedures should result in the same super

parameters, use of $\mathbf{Q}^{1/2}\mathbf{J}$ is normally faster, and a little less prone to numerical error as it involves fewer matrix operations.

If SVDA_EXTSUPER is set to 2, PEST calculates super parameters using the first *m* "**v**" vectors computed by the LSQR algorithm operating on $\mathbf{Q}^{1/2}\mathbf{J}$, where *m* is the number of super parameters cited in the PEST control file. If SVDA_EXTSUPER is set to -2, these vectors are orthogonalised before being employed for definition of super parameters. In cases where the number of base parameters is every high (more than about 2500), use of the LSQR algorithm for super parameter definition will result in much faster computation of these parameters.

If SVDA_EXTSUPER is set to 1 PEST does not calculate super parameters from the base parameter Jacobian matrix at all. Instead it reads from a matrix file the directions in parameter space for which super parameters describe scalar projections. The matrix file format employed by PEST utility programs is described in part II of this manual.

If *m* super parameters are featured in a PEST control file in which SVD-assisted inversion is requested, then the directions in parameter space associated with these parameters are defined as the first *m* columns of the matrix supplied in a matrix file. If more than *m* columns are featured in this matrix, columns after the *m*'th are ignored.

The matrix file supplied to PEST as the super parameter definition file must meet certain requirements. As already stated, it must have as many columns (or more) as there are super parameters defined in the super parameter PEST control file; the names associated with these columns in the matrix file are ignored. The matrix must also have at least as many rows as there are base parameters defined in the base parameter PEST control file. Furthermore each adjustable parameter defined in the base PEST control file should have a matrix row associated with it (and linked to it through equality of base parameter name and associated matrix row name, the latter being provided in the matrix file). It is not essential that row names be supplied in the same order in the matrix file as parameter names in the base parameter PEST control file. Nor does it matter if the matrix file contains extra rows linked, perhaps, to fixed or tied base parameters, or to other parameters altogether. However it is essential that no adjustable base parameter name be unmatched to a row of the matrix provided in the matrix file.

The following should be noted.

1. It is important for the sake of calculation of derivatives with respect to base parameters that all columns of the external super parameter definition matrix have a magnitude of approximately 1. Use of the MATSVD utility described in part II of this manual can ensure this, at the same time as it ensures orthogonality of super parameter vectors. Simply prepare a candidate matrix based, for example, on what are judged to be the most important parameters and/or parameter combinations, and then employ MATSVD to calculate a complementary orthogonal, normalised matrix.
2. If SVDA_EXTSUPER is set to 1 there is no role for the SVDA_SCALADJ variable. Hence it is set to zero inside of PEST.
3. The SVD-assisted inversion process handles base parameter bounds imposition slightly differently when super parameter directions are assigned externally, from the way in which bounds imposition is handled when super parameter directions are calculated by PEST. In the latter case super parameter directions are re-defined by PEST with salient base parameters stuck to their bounds when the latter encounter these bounds; they are thus effectively removed from the inversion process. In the

former case super parameter re-definition cannot take place. As a result, the inversion process may be a little slower, and less efficient, under these circumstances.

SVDA_SUPDERCALC

Under certain circumstances it is possible to eliminate the need to calculate super parameter derivatives by finite differences on the first iteration of an SVD-assisted inversion exercise. In particular, if

1. SVDA_EXTSUPER is set to a value other than 1, indicating that super parameters are calculated by PEST on the basis of a base parameter Jacobian matrix, and
2. this matrix pertains to the same observations and prior information equations as those featured in the super parameter PEST control file,

then derivatives of super parameters can be calculated from base parameter derivatives, this eliminating the need for them to be calculated using finite differences. (Fortunately, these are the most common conditions in which SVD-assisted inversion is undertaken.)

The SVDA_SUPDERCALC control variable activates the functionality whereby PEST is able to dispense with finite-difference calculation of super parameter derivatives during its first iteration, calculating these from base parameter derivatives instead. If SVDA_SUPDERCALC is omitted or set to zero, then super parameter derivatives calculation takes place through finite differences during the first iteration of the SVD-assisted inversion process. However if it is set to 1, then PEST calculates super parameter derivatives internally for the first iteration of the SVD-assisted inversion process, removing the necessity for any model runs to be undertaken in calculating super parameter derivatives during this iteration.

SVDA_PAR_EXCL

This optional SVD-assist control variable is employed when PEST is run in "pareto" model. It is described in chapter 13 of this manual.

10.5 SVD-Assisted Inversion

10.5.1 What PEST Does

Upon commencement of an SVD-assisted parameter estimation run PEST reads the super parameter PEST control file, this being the file provided to it through its command line. It then reads the base PEST parameter control file. Unless SVDA_EXTSUPER is set to 1 signifying user-defined super parameters, it then reads the Jacobian matrix file cited in the "svd assist" section of the super parameter PEST control file, checking that this file and the base parameter PEST control file are compatible. Then, depending on the setting of SVDA_EXTSUPER, it forms either $\mathbf{Q}^{1/2}\mathbf{J}$ or $\mathbf{J}^t\mathbf{Q}\mathbf{J}$, where $\mathbf{J}$ is the base parameter Jacobian matrix and $\mathbf{Q}$ is the observation weight matrix. Note that rows of the base parameter Jacobian matrix pertaining to any prior information present in the base PEST control file are omitted in formulating this matrix. So too are base parameters that are fixed and tied in the original PEST control file; hence these base parameters remain fixed and tied through the SVD-assisted parameter estimation process implemented by PEST.

PEST undertakes singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{J}$ or $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix, retaining only as many singular values as the number of super parameters requested by the user. On the basis of the singular vectors corresponding to the retained singular values it writes a template file

for the PARCALC input file *parcalc.in*; the information in this file allows PARCALC to calculate base parameter values when supplied with current super parameter values by PEST. This template file is re-written at the beginning of every iteration of the inversion process. This is necessary because estimable directions in parameter space are always calculated relative to current parameter values (which change from iteration to iteration). Also, if any base parameters hit their bounds, the singular value decomposition process is repeated, with new singular values and singular vectors being calculated as a consequence.

Thus as PEST alters the values of super parameters, either incrementally for the purpose of derivatives calculation, or in response to the upgrading of these parameters, the values of these super parameters are written to a model input file, as in normal PEST operation. However on this occasion the model input file is *parcalc.in*. PARCALC then transforms these super parameter values into base parameter values, writing all necessary model input files using these base parameters on PEST's behalf. It also writes a PICALC input file on the basis of these base parameters so that PICALC can undertake calculation of prior information equations on PEST's behalf. All model outputs, and all prior information equation results, are then read directly by PEST after each model run.

10.5.2 Best Parameter Estimates

The values of super parameters estimated by PEST through the SVD-assisted parameter estimation process have no inherent meaning. Even within a single parameter estimation run, the calculation of base parameter values from super parameter values changes according to current estimates of the latter parameters.

As has been discussed previously, in accordance with its normal protocol, PEST records current best estimates of super parameter values to a file named *case-svda.par* where *case-svda* is the filename base of the super parameter PEST control file. However when implementing SVD-assisted inversion, PEST records another parameter value file named *case.bpa*, where *case* is the filename base of the base parameter PEST control file. (This is the file used by SVDAPREP to build the control file used for SVD-assisted parameter estimation). At any stage of the parameter estimation process this file contains best estimates of base parameter values.

In normal operation, when the parameter estimation process is complete (or if PEST execution is halted using the PSTOPST command), PEST undertakes a single model run using optimised parameters before terminating execution; thus model input and output files contain best-fit parameter values and corresponding best-fit model outputs. This is not possible when undertaking SVD-assisted parameter estimation. However, based on the contents of the *case.bpa* file, the user can carry out such a model run him/herself. This is accomplished through use of the PARREP utility in the manner described in section 10.1.2.

10.5.3 Parallelised SVD-Assisted Inversion

Parallel PEST and BEOPEST can be used for SVD-assisted parameter estimation, just as the normal PEST can. However a little care should be taken in transferring files to other machines prior to starting Parallel PEST or BEOPEST when undertaken SVD-assisted inversion. The following should be noted.

1. *parcalc.exe* (and *picalc.exe* if prior information is used) should be run-able from all slave machines.

2. If using Parallel PEST for non-SVD-assisted inversion, template files do not need to be transferred to slave machines. The situation is different for SVD-assist; it is now PARCALC, and not PEST, which writes model input files, and PARCALC is part of the model run by the PEST slave. Hence template files of model input files must be transferred to all slave machines so that PARCALC can use them. The same, of course, applies for BEOPEST (for which all template files must be transferred to slave machines regardless of whether or not the inversion process is SVD-assisted – see chapter 11).
3. The new model batch file *svdabatch.bat* must be transferred to all slave machines.
4. If prior information is employed in the SVD-assisted parameter estimation process, then the template file for the PICALC input file (i.e. *picalc.tpl*) must be transferred to all slave machines.
5. If using parallel PEST, then PSLAVE runs the model on slave machines. When supplying PSLAVE with the command to run the model, use "*svdabatch.bat*".
6. If using BEOPEST, then BEOPEST itself, acting as a slave to the BEOPEST master, runs the model. Provide the BEOPEST slave with the name of the super parameter PEST control file on its command line.

10.6 Some Final Points

10.6.1 JCO2JCO and other Utility Programs

The JCO2JCO utility is described in part II of this manual. However it is worth reminding the reader of its existence (and of the existence of some other utility programs) at this point of the present manual.

Suppose that you have undertaken a laborious base parameter run in order to calculate a base parameter Jacobian matrix. Suppose that you then decide to fix certain base parameters, tie certain base parameters to parent parameters, change the log-transformation status of some parameters and/or that you would like to remove some observations from the base parameter PEST control file before calibrating the model. There is no need to undertake the base Jacobian run again. Simply create a new PEST control file with the desired parameters tied, fixed or retransformed, and/or with observations removed, and then use the JCO2JCO utility to create a corresponding JCO file. (Note that if you remove observations you will also need to remove corresponding instructions from instruction files.)

If the PEST control file in which you would like to tie or fix parameters includes regularisation prior information equations generated by the ADDREG1 utility, these can be removed using the SUBREG1 utility. Parameters can then be tied and fixed as required; ADDREG1 can then be used again to generate prior information equations pertaining to the remaining adjustable parameters. JCO2JCO can then be used to create a JCO file for the new, regularised, PEST control file.

10.6.2 Efficiency

Experience has demonstrated that, despite measures available through use of the SVDA_SCALADJ control variable, SVD-assisted inversion can sometimes be slow where some base parameters are log-transformed and others are not. If this is the case then consider log-transforming all parameters in the base PEST control file (but only if the lower bounds of

all base parameters are greater than zero). Once you have done this, the JCO2JCO utility can be used to create a new base PEST control file. If parameters are regularised, consider use of the SUBREG1 and ADDREG1 utilities as appropriate.

10.6.3 Null Space Monte Carlo

SVD-assist, in conjunction with utility programs described in part II of this manual, provides a mechanism for generation of many different random parameter fields which all satisfy the constraints of expert knowledge at the same time as they all satisfy the need for model outputs to fit the calibration dataset.

Suppose that you have just calibrated a model. Suppose also that the dimensions of the null space are large. Supposedly, the calibrated parameter field is of minimum error variance precisely because it includes no null space components. See Doherty (2015) for a discussion of this important issue.

Using the PARREP utility, generate a new base parameter PEST control file whose initial parameter values are calibrated parameter values. Remove regularisation from this file (using the SUBREG1 utility if desired.) Set NOPTMAX to -2 in this file and generate a new Jacobian matrix file based on these best-fit parameters.

Now use the RANDPAR utility in conjunction with the prior parameter covariance matrix to generate many different parameter fields centred on the calibrated parameter set. Next, use the PNULPAR utility to replace the solution space components of these random fields with that of the calibrated model. The resulting set of random parameter fields would allow the model to fit the calibration dataset immediately if the model were linear. However, because the model is probably not linear, the fit with the calibration dataset will probably not be as good as that achieved through the original calibration process.

Use SVDAPREP to construct an SVD-assisted PEST control file based on the one which was just used to generate the JCO file. The "svd assist" section of this file "points back" to the base PEST control file (which contains the calibrated parameter set). In figure 10.2 the base PEST control file is named *case.pst* while the SVD-assist PEST control file is named *case-svda.pst*. If PEST is run using *case-svda.pst* super parameters are constructed based on observations and weights cited in *case.pst* and sensitivities recorded in *case.jco*. As was described above, if a calibration exercise is then undertaken, the super parameter Jacobian matrix for the first iteration of the inversion process is generated using sensitivities recorded in file *case.jco*. Only if the inversion process requires a second iteration does a super parameter Jacobian matrix need to be calculated using finite parameter differences.

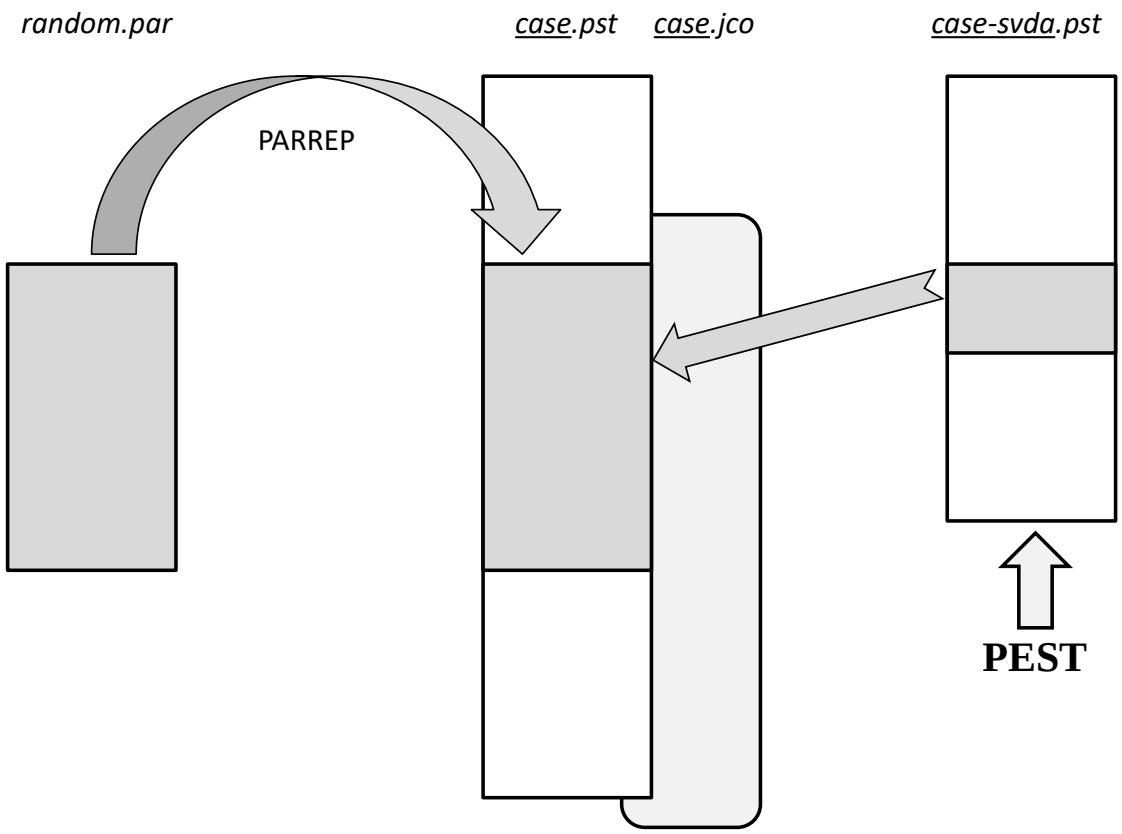


Figure 10.2 The role of SVD-assist in the null space Monte Carlo process.

There is, of course, no need to re-calibrate an already calibrated model. However if PARREP is used to replace the calibrated parameter set residing in file *case.pst* with a random parameter set which has been null space projected using PNULPAR as described above so that it almost calibrates the model, then this random parameter field can be adjusted to satisfy calibrated constraints by running *case-svda.pst*. As usual, super parameters are constructed using observations and weights from *case.pst* and sensitivities from *case.jco*. The initial iteration of the inversion process uses sensitivities recorded in file *case.jco* to calculate super parameter sensitivities, so finite-difference derivatives calculation is not needed for this iteration. Furthermore the initial random parameter values residing in *case.pst* have been modified by PNULPAR; hence they do not de-calibrate the model much. Only one iteration (with sensitivities provided "for free") may be required for adjustment of these parameters to meet calibration constraints. If a second iteration is needed, only as many model runs must be undertaken as there are dimensions in the solution space; that is, only as many model runs are required as there are super parameters cited in file *case-svda.pst*. Through use of the NOPTMAX, PHISTOPTHRESH, PHIABANDON and LASTRUN control variables you can limit the number of model runs that you are prepared to commit to any calibration exercise undertaken in this manner.

The above methodology can be varied in different circumstances. Instead of using RANDPAR to generate random parameter fields based on the prior parameter covariance matrix, and then using PNULPAR-based null space projection to promote respect of these parameter fields for calibration constraints, RANDPAR can directly sample a linear approximation to the posterior parameter covariance matrix calculated using the PREDUNC7 utility (see part II of this manual). Parameter sets thus-generated should be centred on the calibrated parameter set. If the null space is not large, then these samples of the linearized

posterior parameter distribution can be adjusted so that they respect calibration constraints by running PEST repeatedly (with a new RANDPAR-generated initial parameter set each time) with the "/i" switch. Sensitivities in *case.jco* are thus used for the first iteration of the random parameter adjustment process, this saving the need for finite-difference derivatives calculation during the first iteration of the calibration process. Alternatively, these samples of the linear approximation to the posterior parameter distribution can be sequentially PARREPed into *case.pst* in the manner depicted if figure 10.2; PEST can then be run using *case-svda.pst* to enforce calibration constraints on these parameter fields. As a third alternative, these samples of the linear approximation to the posterior parameter distribution can be null-space-projected using the PNULPAR utility. Because null space projection follows sampling of a linear approximation to the posterior parameter distribution, it is most unlikely that the resulting random parameter fields will require any more than minor adjustment to satisfy calibration constraints.

See section 8.2 of Doherty (2015) for more discussion of all of these alternatives.

11. Parallel PEST and BEOPEST

11.1 General

11.1.1 Introduction

In the course of calibrating a model PEST must run that model many times. Some model runs are made in order to test parameter upgrades calculated using different Marquardt lambdas. Others are made with certain parameters temporarily incremented as part of the process of calculating the Jacobian matrix. In calculating the Jacobian matrix PEST needs to run the model at least as many times as there are adjustable parameters (more than this if higher order finite differences are employed). In most cases by far, the bulk of PEST's time is consumed in running the model. It follows that any time savings that are made in carrying out model runs will be reflected in dramatic reductions in PEST's overall run time.

Parallelisation of model runs can achieve such enhanced PEST performance. Model runs can be parallelised over different cores belonging to the same machine, over different machines that are part of a local network, over machines that are connected through the internet, or over virtual machines on the cloud. Through parallelisation of model runs, overall PEST run times can be reduced by a factor almost equal to the number of machines over which model runs can be distributed.

The inversion (including SVD assist) and predictive uncertainty analysis algorithms implemented by Parallel PEST and BEOPEST are no different from those implemented by the normal PEST. Preparation of template files, instruction files and the PEST control file is identical to that of the normal PEST. However use of Parallel PEST requires that one extra file be prepared prior to undertaking an inversion run, namely a "run management file". This file informs Parallel PEST of the machines to which it has access, the names of model input and output files residing on those machines, and the name of subdirectories (i.e. folders) that it can use on each of these machines to communicate with one or more "slaves" which carry out model runs on those machines. In contrast, for BEOPEST, the run management file is optional.

While parallel PEST and BEOPEST employ similar run management algorithms, and identical inversion engines, the difference between them is in how the "master program" which implements the inversion process communicates with the "slaves" which carry out model runs. Parallel PEST uses message files for communication between master and slaves; BEOPEST uses TCP/IP. The latter means of communication is faster, and does not require that the master have read/write access to a slave's working directory. Hence the BEOPEST master and its slaves can be on other sides of the world.

11.1.2 Parallelisation of the Jacobian Matrix Calculation Process

When PEST calculates derivatives of model outcomes with respect to adjustable parameters using finite parameter differences, successive model runs are completely independent; that is, the parameters used for one particular model run do not depend on the results of a previous model run. The complete independence of model runs undertaken for the purpose of filling the Jacobian matrix makes this process easily parallelised. Under these circumstances Parallel PEST and BEOPEST simply distribute model runs to different machines or processors as they become available, and read the outcomes of these runs as they are finished.

11.1.3 Parallelisation of the Parameter Upgrade Process

Unlike the Jacobian calculation process, the process of testing parameter upgrades calculated on the basis of different Marquardt lambdas is difficult to parallelise. This is because the search for a Marquardt lambda that achieves the best set of parameters (with "best" depending on the current mode of PEST's operation) is an inherently serial procedure. The choice of what lambda to use next depends on the outcome of the model run undertaken using the previously-chosen lambda.

Nevertheless, the lambda-testing procedure can be accelerated to some degree through "partial parallelisation" of this procedure. Recall from section 4.2.6 that tested values of the Marquardt lambda are related to each other through a factor of RLAMFAC (a variable supplied in the "control data" section of the PEST control file). Parallel PEST and BEOPEST commission simultaneous model runs across the different processors to which they have access using parameter upgrades calculated on the basis of a series of lambda values related to each other by a factor of RLAMFAC. Some of these runs may be "wasted"; serial testing of Marquardt lambdas may quickly establish the optimality of the first or second in a series of thus tested Marquardt lambdas. However the opposite may also occur. It is a greater waste of computing resources for processors to lie idle when they could be otherwise usefully engaged in pre-emptive testing of a wide range of lambda values then for the outcomes of some of these runs to go unused.

A number of settings are available to the user that control the way in which the partial parallelisation process operates. Regardless of these settings, parallel lambda runs will not be undertaken if PEST is running in "predictive analysis" mode and a line search is undertaken as part of the predictive analysis process.

Depending on user-supplied settings, parallelisation of the Marquardt lambda testing procedure may be disallowed if parameters are frozen at their upper or lower bounds. The rationale behind serialization of the lambda testing procedure under these conditions is that PEST may not be able to predict which parameters will remain at their bounds if others are frozen there and, if they don't, the order in which they should be progressively unfrozen and the inverse problem reformulated to take account of this.

11.1.4 Communication Overheads

If model run times are short, gains in computational efficiency that are achievable using Parallel PEST will not be as great as when model times are large, for the time taken in writing and reading (possibly lengthy) model input and output files across a local area network may then become large in comparison with model run times. Another contributor to parallelisation latency is the use of message files for communication between master and slaves. It may take a number of seconds before a message issued by one is received by the other.

Because it uses TCP/IP for messaging, and because the BEOPEST slave writes model input files and reads model output files locally, master/slave communication overheads are very low for BEOPEST.

11.1.5 Installing Parallel PEST and BEOPEST

The command line version of the Parallel PEST executable *ppest.exe*, and its 64 bit version *i64ppest.exe*, are automatically installed when you install PEST on your machine. The same applies to the BEOPEST executables *beopest32.exe* and *beopest64.exe*.

As is explained below, for Parallel PEST to run a model on another machine it must signal a slave, named PSLAVE, residing on the other machine to generate the command to run the model. Thus *pslave.exe* must be installed on each machine engaged in the Parallel PEST parameter estimation process. To do this, copy *pslave.exe* (also provided with PEST) to an appropriate directory (i.e. folder) on each such machine. This folder can be the model working folder on that machine if desired; if not, it should be a folder whose name is cited in the PATH environment variable on that machine.

BEOPEST does not use a special slave program. Rather BEOPEST itself is the slave. Hence *beopest32.exe* or *beopest64.exe* must be copied to slave machines in the same way as was described above for *pslave.exe*.

As will be discussed below, not only PEST programs, but model programs and accompanying files must also be copied to slave machines prior to initiating a parallel PEST or BEOPEST run.

11.1.6 The Parallel Run Queue

Parameter values corresponding to queued parallel model runs, and model output values calculated on the basis of completed parallelized model runs, are not stored internally by Parallel PEST or BEOPEST. Instead they are stored in binary, direct access files. This reduces PEST memory requirements considerably. These direct access files are named *pest###.dap* (for parameter values) and *pest###.dao* (for model output values). These files are normally deleted at the end of a PEST run. However if PEST is stopped prematurely, they may still be present in the Parallel PEST or BEOPEST working folder. If execution of Parallel PEST or BEOPEST is re-started with the "/s" restart switch, then these files are still needed, for they allow Parallel PEST or BEOPEST to resume their processing at the same place at which their previous run was terminated. If you have no intention of restarting PEST, then feel free to delete these files yourself as PEST will delete them anyway at the commencement of its next run if that run is initiated in that same working folder without the "/s" switch.

Parallel PEST is described next. BEOPEST is described after that.

11.2 Parallel PEST – Concepts and Specifications

11.2.1 Model Input and Output Files

The manner in which Parallel PEST carries out model runs on different machines is just a simple extension of the manner in which PEST carries out model runs on a single machine. Before running a model on any machine, Parallel PEST writes one or more input files for that model, these files containing parameter values appropriate to that model run. After the model has finished execution, Parallel PEST reads one or more files generated by the model in order to obtain values calculated by the model for a set of outcomes for which there are corresponding field or laboratory measurements.

Operation of Parallel PEST assumes that PEST can write model input files and read model output files, even though these files may reside on a different machine to that on which PEST itself is running. Input and output files for a particular model may reside on the machine which actually runs the model, or on a network server to which both PEST and the model's machine have access. The only provisos on where these files reside is that both PEST and the model must be able to read and write to these files, and that these files are named using

normal filename protocol. This is easily accomplished through the use of the "shared folder" concept available across local area networks.

Parallel PEST writes input file(s) for the models running on the various networked machines using one or more templates residing on the machine on which PEST is running. Similarly, Parallel PEST reads the output file(s) produced by the various models using the instructions contained in one or a number of instruction files residing on the machine on which PEST runs.

11.2.2 The Parallel PEST Slave Program

While Parallel PEST is able to achieve access to model input and output files residing on other machines through the use of shared folders, it cannot actually run the model on another machine; only a program running on the other machine can do that. Hence before PEST commences execution, a "slave" program must be started on each machine on which the model will run. Whenever PEST wishes to run a model in a particular folder on a particular machine it signals the slave running in that folder on that machine to start the model. Once the model has finished execution the slave signals PEST that the model run is complete. PEST can then read the output file(s) generated by the model.

The slave program (named PSLAVE) must be started before Parallel PEST in each folder on each machine on which model runs are to be undertaken. It detects the commencement of PEST execution through reading a signal sent by PEST as the latter starts up. It then awaits an order by PEST to commence a model run, upon the arrival of which it sends a command to its local system to start the model. It is possible that the command used to start the model may be different for different slave machines (for example if the model executable resides in a differently-named folder on each such machine and the full model pathname is used in issuing the system command); hence PSLAVE prompts the user for the local model command as it commences execution.

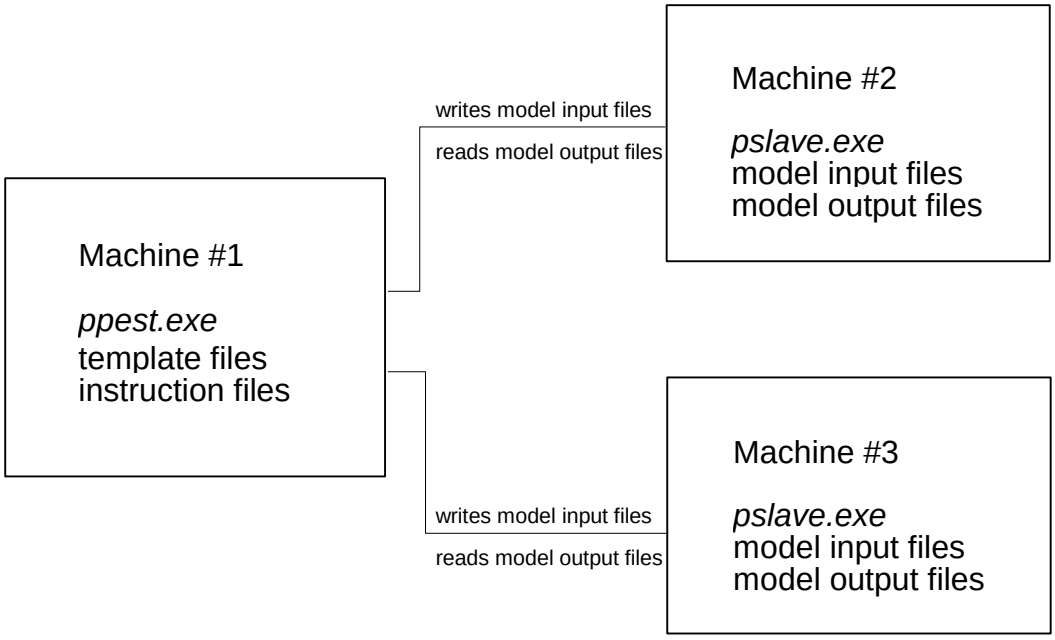


Figure 11.1 Relationship between PEST, PSLAVE and the model.

Figure 11.1 illustrates in diagrammatic form the relationship between Parallel PEST, PSLAVE and the model for the case where PEST resides on one machine and the model is

run on each of two other machines. Note that, as is explained below, this is an unusual case in that it is common practice for the master machine (i.e. machine #1 in figure 11.1) to also be a slave machine to avoid wastage of system resources.

11.2.3 Running the Model on Different Machines

Greatest efficiency can be achieved if an independent model executable program resides on each slave machine. Thus when PSLAVE runs the model, the executable program does not have to be loaded across the network. Note however, that if PEST and multiple incidences of PSLAVE are being run on the same machine in order to gain access to multiple processors belonging to that machine, there is no reason why each slave should not run the same model executable.

It is essential that for each slave engaged in the Parallel PEST inversion process, the model as run by that slave reads its input file(s) and writes its output file(s) to a different folder from the model as run by any of the other slaves. If this is not done, model output files generated during one parallel model run will be overwritten by those generated during another; similarly model input files prepared by PEST for one particular model run will be overwritten by those that it prepares for another.

In many cases of model usage all input files for one particular model are housed in a single folder; also all model output files are written to this same folder. In this simple case, preparation for model runs on different machines across a network consists in simply copying the entire model working folder from the master machine to appropriate folders on each of the other machines. As is explained below, the structure of the "run management file" which Parallel PEST must read in order to ascertain the whereabouts of each of its slaves is particularly simple under these circumstances.

11.2.4 Communications between Parallel PEST and its Slaves

Parallel PEST must communicate with each of its slaves many times during the course of an inversion process. It must inform each slave that it has begun execution, it must command various slaves to run the model, and it must receive signals from its slaves informing it that different model runs have reached completion. It must also inform all slaves to shut down under some circumstances of run termination, and be informed by each slave, when it commences execution, that the slave itself is up and running.

Such signalling is achieved through the use of short shared "signal" files. These files, whether originating from PEST or PSLAVE, are written to the folder from which PSLAVE is run on each slave machine; PSLAVE provides these signal files with no path prefix, thus ensuring that they are written to the folder from which its execution was initiated. PEST however must be informed of the names of these various PSLAVE working folders (most of which will reside on other machines) through its run management file. Note that there is no reason why a PSLAVE working folder should not also be a model working folder on any slave machine, as long as filename conflicts are avoided. In fact such an arrangement, being simpler, reduces the chance of making mistakes. Table 11.1 shows the names of the signal files used by Parallel PEST and PSLAVE to communicate with each other.

If Parallel PEST carries out multiple simultaneous model runs on the one machine (to make use of multiple processors on that machine), then a separate PSLAVE working folder must be commissioned for each separate PSLAVE running on that machine. Once again, these folders may also be the working folders for each of the distinct model runs.

File	Written By	Function
pslave.rdy	PSLAVE	Informs PEST that it has begun execution; also informs it of the command line which it will use to run the model.
pest.rdy	PEST	Informs PSLAVE that it has begun execution.
param.rdy	PEST	Informs PSLAVE that it has just generated model input file(s) on the basis of a certain parameter set and that it must now run the model.
observ.rdy	PSLAVE	Informs PEST that the model has finished execution and that it must now read the model output files(s).
pslave.fin	PEST	Informs PSLAVE that it must now cease execution.
p###.##	PEST	Used to test whether PEST has access to all PSLAVE working folders.

Table 11.1 Files used by PEST and PSLAVE to communicate with each other.

Where Parallel PEST is used to run the model in different folders on different machines across a network, it is likely that one such folder in one such slave machine will be that in which the PEST master is running. (Note that the PEST master is not, in general, a big consumer of system resources, much of its role being to manage input and output to and from the model runs being initiated by its various slaves.) Running PSLAVE from the model working folder, thus designating it as a PSLAVE working folder, is entirely appropriate. This folder can also be that from which PEST is run and may thus hold PEST control, template and instruction files. See below for a further discussion of this topic.

11.2.5 The Parallel PEST Run Management File

The purpose of the Parallel PEST run management file is to inform PEST of the working folder of each slave (as seen through the network from the machine on which the PEST master is run), and of the names and paths pertaining to each model input file which it must write and each model output file which it must read. The run management file must possess the same filename base as the current PEST case; its extension must be "*.rmf*". Thus if Parallel PEST is run using the command

```
ppest case
```

then PEST will look to file *case.pst* to read its control data and file *case.rmf* to read data pertaining to the distribution of model runs across the network.

Figure 11.2 shows the structure of a run management file; as usual, control variables that are surrounded by square brackets are optional. Figure 11.3 shows an example of a run management file for a case where there are three slaves.

```
prf
NSLAVE IFLETYP WAIT PARLAM [RUNREPEAT] [run_slow_fac=x]
SLAVNAME SLAVDIR [SLAVEGROUP]
(once for each slave)
(RUNTIME(I), I=1,NSLAVE)
Any lines after this are required only if IFLETYP is nonzero; the following
group of lines is repeated once for each slave.
INFLE(1)
INFLE(2)
(to NTPFLE lines, where NTPFLE is the number of template files)
OUTFLE(1)
OUTFLE(2)
(to NINSFLE lines, where NINSFLE is the number of instruction files)
```

Figure 11.2 Structure of the Parallel PEST run management file.

```
prf
3 1 1.5 0
'my machine' .\
'steve''s machine' k:.\
'jerome''s machine' l:\model\
600 600 720
model.in1
model.in2
model.o1
model.o2
model.o3
k:.\model.in1
k:.\model.in2
k:.\model.o1
k:.\model.o2
k:.\model.o3
l:\model\model.in1
l:\model\model.in2
l:\model\model.o1
l:\model\model.o2
l:\model\model.o3
```

Figure11.3. A typical Parallel PEST run management file.

The first line of a run management file should contain only the character string "prf" identifying the file as a PEST run management file. The next line must contain four items, and a possible fifth item.

NSLAVE

The first variable on the second line of a PEST run management file is the integer variable NSLAVE. This is the number of slaves involved in the current Parallel PEST run.

IFLETYP

The second item on the second line of the run management file (an integer variable) is IFLETYP; this must be supplied as either 1 or 0. If it is 1 then all model input and output files on the various slave machines must be individually named (as is demonstrated in figure 11.3). However if the names of all corresponding input and output files are the same for all slaves, being identical to the names provided (without a folder name) in the "model input/output" section of the PEST control file, and if each set of model input and output files lies within each PSLAVE working folder on each slave machine, then a value of 0 can be supplied for IFLETYP and the model input and output filenames omitted from the run management file;

Figure 11.5 shows such a run management file. In this case PEST automatically affixes the working folder of each slave to the front of each model input and output file as named in the "model input/output" section of the PEST control file.

WAIT

The third item on the second line of the PEST run management file is the value for the real-valued variable WAIT. As PEST and PSLAVE exchange information with each other, and as PEST writes and reads model input and output files, both PEST and PSLAVE pause at certain strategic places in the communications process for an amount of time specified as the value of the variable WAIT; such pauses allow one machine to "catch up" in implementing the instructions supplied by another machine. WAIT is the duration of each such pause, expressed in seconds; its value must be greater than zero. Experience has demonstrated that if PEST and all of its slaves are running on the same machine a value of 0.2 seconds is normally adequate. However for communications across a busy network, values as high as 10.0 seconds (or even higher) may be appropriate. By pausing at appropriate moments in the data exchange process, "sharing violation" errors can be avoided. In some cases such errors will result in nothing more than a message (generated by the operating system) to the PEST or PSLAVE screen; this matters little as both PEST and PSLAVE check that their instructions have been obeyed, re-issuing them if they have not. However if a model, rather than PEST or PSLAVE encounters such an error through reading from or writing to a file which has not been closed by another process (such as PEST or even the previous model run), then the operating system will issue a message such as

```
Sharing violation reading drive C
Abort, Retry, Fail? [y/n]
```

The slave running that particular model will drop out of the Parallel PEST inversion process until this question is answered. It would obviously be unfortunate if this question is asked at midnight when no-one is around to answer it with a simple "r" to send the model on its way again. Fortunately, if WAIT is set high enough, this should not happen.

PARLAM

If the integer variable PARLAM (the fourth variable appearing on the second line of the run management file) is set to 1, partial parallelisation of the lambda search is undertaken. However if it is set to 0, then the lambda search is conducted in serial fashion using just one processor. Partial parallelisation, and other possible PARLAM settings, are further discussed below.

RUNREPEAT

The optional fifth item on the second line of the Parallel PEST run management file is the integer RUNREPEAT variable. If Parallel PEST encounters a problem in reading a model output file from a slave's folder, it tries a number of times to read the file before giving up. Then, just in case the problem originated in network congestion or some other troublesome network behaviour, Parallel PEST repeats the model run on the same or another slave. A number of repetitions are attempted before PEST terminates execution with an error message to the screen outlining the nature of the problem encountered. Where model run times are long and the problem did not in fact originate in network communication failure, this process can take a long time. There are instances where the user would like to be made aware more quickly of such a problem so that he/she can take steps to rectify it if, in fact, the source of

the problem is in the model, or one of the components thereof, rather than in the network. If RUNREPEAT is set to 0, attempts at model run repetition will not take place. If it is set to any other number (or if it is omitted), attempted run repetition will take place.

RUN_SLOW_FAC

An important task in parallel run management is that of deciding when a model run is overdue. If it is overdue to too great an extent, the Parallel PEST run manager assigns that model run to another slave if there is a slave with nothing else to do, and if PEST judges that the slave will be able to complete this run relatively quickly. The run manager does this because it acts under the assumption that the slave computer has become overloaded with other tasks, or that it may have dropped out of the network altogether.

Sometimes there are other reasons for which a model run may be slow. For example, if model runs are being undertaken for the purpose of testing the efficacy of different Marquardt lambdas in lowering the objective function, the model may experience convergence difficulties when using some updated parameter values. In this case it is not good run management practice for PEST to assign the late model run to a slave that may have finished its model run quickly; it too will then become bogged down in trying to run the model using the same recalcitrant parameter set.

The user can set a factor which instructs the run manager when to deem a model run as overdue. If, for example, this factor is set to 20, then a model run will not be construed as being overdue unless a time interval equal to 20 times the model run time on the pertinent slave has elapsed since the model run was commenced. This variable is named "RUN_SLOW_FAC". It is a real-valued, optional variable whose default value is 1.5. Obviously, this number must be given a value which is greater than 1.0.

The value for RUN_SLOW_FAC is read from the second line of the run management file. A value for RUN_SLOW_FAC is provided by adding the string "run_slow_fac = *x*" to any part of this line. Here *x* is the factor discussed above. Figure 11.4 depicts a run management file in which RUN_SLOW_FAC is set to 5.0.

```
prf
2  0   0.20000  -2  1  run_slow_fac = 5.0
Pest_slave_1  .\
Pest_slave_2  .\test1
1.0 1.0 1.0 1.0 1.0
```

Figure 11.4 A run management file in which the value of RUN_SLOW_FAC is set to 5.0.

Remainder of Run Management File

Lines 3 to NSLAVE+2 (i.e. NSLAVE lines in all) of the run management file should contain two entries each. The first is the name of each slave; any name of up to 30 characters in length can be provided. The name should be surrounded by single quotes if it contains any blank spaces; an apostrophe should be written twice. This name is used for identification purposes only; it need bear no resemblance to computer names or IP addresses.

The second item on each of these lines (i.e. SLAVDIR) is the name of the PSLAVE working directory (i.e. folder) as seen by PEST. This folder name should terminate with a backslash character (forward slash on a UNIX system); if you do not terminate the name with a slash, PEST will add it automatically. You can either provide the full path to the PSLAVE working

folder, or supply it in an abbreviated form if this works. Thus if, having opened a command line window to run the Parallel PEST master, you transfer to drive K in figure 11.3 (this being a slave machine's drive) and change your working directory to the PSLAVE working folder, and then change back to the local folder again (for example, by typing "C:"), then a path designation of "K:.\" affixed to the filenames listed in table 11.1 will ensure that these files are written correctly to the PSLAVE working folder regardless of its full pathname. Note however that the folder "K:\" (without the dot) is not the same folder as "K:.\" (with the dot) if folders under the PSLAVE working directory are accessible from the machine on which PEST is run. If there are any doubts, provide the full PSLAVE path.

Note also from figure 11.3 that a slave on the local machine can work from the same folder as the Parallel PEST master. This may be desirable if all model input files are in this same folder (which is often the case) and this is also the current PEST working folder. A designation of "**.** \" is sufficient to identify this folder.

The next line of the run management file must contain as many entries as there are slaves. Each entry is the expected run time of the model on the respective slave, the ordering of slaves on this line being the same as that in which slave data was supplied earlier in the run management file. Run times should be supplied in seconds; if you are unsure of the exact run times, simply provide the correct run time ratios. There is no need for stopwatch precision here as PEST continually updates model run time estimates in the course of the inversion process. However it is better to overestimate, rather than underestimate these run times so that PEST will not re-instigate the initial model run (which is not part of a Jacobian calculation) on an alternative machine if the initial model run takes much longer than you have indicated, and PEST comes to the conclusion that a mishap may have occurred on the machine to which that run was initially assigned.

If the value supplied for IFLETYP is 0, then the run management file is complete. However if it is supplied as 1, the names of all model input files and all model output files in all slave working folders must next be supplied individually. Either full pathnames can be supplied or abbreviated pathnames, the abbreviations being sufficient for PEST to write and read the respective files from the folder in which it is run. Data for the various slaves must be supplied in the same order as that in which the slaves were introduced earlier in the file. For each slave the names of NTPFLE model input files must be followed by the names of NINSFLE model output files, where NTPFLE is the number of template files and NINSFLE is the number of instruction files pertaining to the present PEST case. (Note that the PEST template and instruction file corresponding to each of these model input and output files is identified in the "model input/output" section of the PEST control file read by Parallel PEST.)

Figure 11.5 shows a Parallel PEST run management file equivalent to that of figure 11.3 but with the value of NFLETYP set to 0. Use of an abbreviated run management file such as that shown in figure 11.5 is possible where all model input and output files used by each slave reside in the one folder, and this folder is also the PSLAVE working folder.

```
Prf
3 0 1.5  0
'my machine' .\
'steve''s machine' k:.\
'jerome''s machine' l:\model
600 600 720
```

Figure 11.5 A Parallel PEST run management file equivalent to that of figure 11.3 but with NFLETYP set to zero.

11.2.6 Alternative PARLAM Settings

The algorithm used by Parallel PEST to undertake parallel model runs as part of its lambda testing procedure if the PARLAM control variable is set to 1 bears some resemblance to that used for parallelisation of the Jacobian matrix calculation process. However there are some important differences. One such difference is that if any slave carries out model runs with a run time which is greater than 1.8 times that of the fastest slave, then that slave is not used in the partial parallelisation process. This is because PEST sends model runs to its slaves in "run packets"; it will not resume its normal execution until all runs in the packet are completed. The size of a "run packet" at any stage of the lambda testing procedure is equal to the number of available slaves whose execution speed is roughly equivalent to that of the fastest slave. The "packet" is limited to this size because if one particular slave can complete two model runs in the same or less time than that required for another slave to complete only one model run, then it would be more efficient to undertake these model runs in serial, with the proper decision-making process taking place after each such run.

The principle difference between parallelised Jacobian runs and parallelised lambda testing runs is that in the former case PEST knows the number of runs that must be carried out before the Jacobian calculation process is complete. In contrast, the lambda testing procedure is deemed to be complete when PEST judges that the overall parameter estimation process is better served by terminating the current lambda testing procedure and moving on to the next iteration of the inversion process. The criteria by which this decision is made are supplied through the variables appearing on the fifth line of the "control data" section of the PEST control file. Hence the size of the "packet" of parallel model runs ordered by PEST is determined on the basis of the number and speed of available slaves, and not on the basis of foreknowledge of the number of parallel runs required for completion of the lambda testing procedure, for this knowledge is not available. The decision-making process involved in the lambda testing procedure is activated only after each packet of model runs is complete, a process that may result in some of these runs being ignored. The fact that the lambda adjustment procedure then becomes a combination of parallelisation with intermittent decision-making is the basis for its classification as a "partial parallelisation" procedure.

During any iteration of the inversion process, upon commencement of the lambda testing procedure for that iteration, PEST's first packet of model runs is based on Marquardt lambdas which are generally lower than the optimal lambda determined during the previous iteration. However, if there are enough slaves at its disposal, PEST also initiates model runs based on one or a number of higher Marquardt lambda values as well. On subsequent occasions during the same lambda testing procedure on which PEST orders packets of model runs to be completed, parameters used for such runs are all calculated on the basis of decreasing Marquardt lambdas or on increasing Marquardt lambdas, depending on the results of the previous package of parallel runs.

The lambda testing procedure is such that parallelisation inevitably results in some model runs being wasted. Hence, although PEST might inform the user through its screen output that *n* parallel model runs are being carried out, it may not display the results (i.e. the objective function and perhaps the model prediction) of all of these *n* model runs. It simply processes the results of that "packet" of runs in accordance with a lambda testing algorithm that is similar (but not identical) to that which would be used if these lambda testing runs had not been parallelised at all. If the demands of that algorithm are such that more Marquardt lambdas must then be tested, another "packet" of runs is initiated.

Nevertheless, there will be some occasions on which the path taken by the inversion process

is slightly different when the lambda search procedure is parallelised from that which would be taken if the lambda search is conducted on the basis of serial model runs. This will occur if an unexpected and significant advance in the inversion process is achieved in a run that would not have been undertaken on the basis of the usual Marquardt lambda testing procedure based on serial model runs.

Where the number of slaves involved in the parallelisation process is high, it is wise to limit the number of slaves which actually become involved in the Marquardt lambda testing procedure. One reason for this is that, with so many different lambdas being simultaneously tested, it is possible for some to yield upgrade vectors containing parameters which create problems for the model. In general, the further is a tested lambda from the current lambda, the more likely is this to happen. Problems could include slower model convergence or even model failure.

In response to this imperative, PARLAM can be set to a negative number. As long as it is non-zero, partial parallelisation of the lambda search procedure is activated. However the number of slaves involved in this lambda search procedure is limited to the absolute value of PARLAM. Thus, for example, if parallelisation is being undertaken using 50 slaves, and if PARLAM is set to -4, then Parallel PEST involves only 4 slaves in the partial parallelisation of lambda testing.

A PARLAM setting of -9999 however has special significance. This setting should only be employed where a user has access to a relatively large number of computing nodes of equal power. If PARLAM is set to -9999, the parallel lambda testing procedure follows the strategy set out below.

1. Only one cycle of parallel runs is devoted to testing Marquardt lambdas; PEST will not commit to a second cycle, irrespective of results forthcoming from the first cycle.
2. The maximum number of parallel runs constituting this cycle is set to NUMLAM, the variable in the "control data" section of the PEST control file which sets the maximum number of lambda-testing runs undertaken per iteration. However if there are less slaves than NUMLAM available at the time that lambda testing is required, NUMLAM is temporarily reduced to the number of available slaves.
3. If it is not supplied as a negative number, RLAMFAC (the lambda multiplier employed in lambda testing) is set internally to the negative of its supplied value. Hence, as described in section 4.2.6 of this manual, a much wider range of lambda values is tested than would be the case if a value-independent multiplier were employed.
4. Tested lambda values (on different slave machines) are disposed in equal numbers above and below the current value of the Marquardt lambda.
5. The outcomes of all such runs are read, irrespective of "trends" discovered from the processing of early returns. Thus if the dependence of the objective function on the Marquardt lambda is irregular (such as can occur if finite-difference derivatives are compromised as a result of model output granularity), an "accidental" low objective function corresponding to a random lambda will not be missed.

Use of a PARLAM setting of -9999 is recommended where a user has access to a large number of computing nodes, many of which would be standing idle while implementing a partially parallelised lambda testing strategy. The user must remember however, that the number of model runs employed for lambda testing is set by NUMLAM, and not by the

number of slaves/nodes available. Hence NUMLAM may need to be set higher than normal. Note also that, as stated above, only one round of lambda-testing runs is undertaken.

It is important to note that for all PARLAM settings other than -9999 Parallel PEST will serialize the lambda testing procedure if any parameter hits its bound, even though it may have commenced the lambda testing procedure as a parallelised process. As stated above, this is because it reformulates the inverse problem to accommodate the fact that the parameter is temporarily non-adjustable.

Despite its advantages, caution should be exercised when setting PARLAM to -9999. If a model is susceptible to crashing when it is supplied with parameter values that it finds unpalatable, this is most likely to occur during the lambda testing procedure, for this is the part of the parameter estimation process where parameter values are most different from those that have been employed previously. Testing of a wide range of lambda values in this parallel fashion can increase the chances of model run failure. Note, however, that PEST failure can be prevented through an appropriate setting of the LAMFORGIVE control variable. This can reduce some of the risks associated with setting PARLAM to -9999.

Another negative feature of a -9999 PARLAM setting is that PEST's ability to sequentially and temporarily freeze parameters that have hit their bounds (and re-formulate the inverse problem accordingly) is compromised to some extent. Nevertheless the methodology for partially parallelised lambda testing supported by a PARLAM value of -9999 can be of great use where, as stated above, model derivatives are compromised because of model numerical imperfections, and/or the number of processors to which Parallel PEST has access is high.

If setting PARLAM to -9999, it is a good idea to set RLAMFAC to a higher negative value than normal (for example a value of -4 rather than the recommended values of -2 or -3) so that it can sample the objective function vs. lambda curve in greater detail (perchance to fall into objective function crevices in a possibly noisy relationship between Marquardt lambda and the objective function).

11.2.7 PARLAM Override

If NUMLAM is set to a negative number in the PEST control file, and if PEST is run as Parallel PEST, the PARLAM variable is automatically set to -9999. The number of parallel model runs used in the Marquardt lambda testing procedure is then equal to the absolute value of NUMLAM. For Parallel PEST a negative NUMLAM value thus overrides the PARLAM value provided in the run management file.

11.2.8 Slave Groups

Parallel PEST optionally allows slaves to be gathered into groups. If this is done, the name of the group to which each slave is assigned must follow the name of its working directory in the Parallel PEST run management file. Figure 11.6 shows an example.

```
prf
3  0   0.10000    -2 0
Pest_slave_1  .\test1   "group1"
Pest_slave_2  .\test2   "group1"
Pest_slave_3  .\test3   "group2"
1.0 1.0 1.0 1.0
```

Figure 11.6 Example of a Parallel PEST run management file showing slave groups.

The following rules must be followed when assigning slaves to groups.

1. If any slave is allocated to a group, all slaves must be allocated to a group.
2. If a group name contains a space, this name must be surrounded by quotes when supplied in the run management file. If not, quotes are optional.
3. Slave group names are case-insensitive; these names are converted to lower case internally.

The group concept is used to allocate runs to slaves. When undertaking model runs on the basis of different Marquardt lambdas, the number of runs that must be undertaken through any run packet may be considerably less than the number of slaves at PEST's disposal. The reasons for this have been discussed above. In this case, PEST attempts to allocate runs preferentially to slaves belonging to different groups; it will not allocate two model runs to a particular slave group as long as all slaves belonging to another group are idle. Thus, for example, where slaves are designed to use different nodes on a relatively small number of different machines, and are subdivided into groups on the basis of the machine to which each node belongs, this strategy minimises the run load on any single machine. Hence if slaves are allocated to machine-specific groups, then if a machine is available on which no model run is taking place, this will be allocated a model run instead of a machine on which a run is already taking place.

11.3 Using Parallel PEST

11.3.1 Preparing for a Parallel PEST Run

Before running Parallel PEST, a PEST run should be set up in the usual manner. This will involve preparation of a set of template and instruction files as well as a PEST control file. When preparing the PEST control file it is important for template and instruction files to be properly identified. However the names of the corresponding model input and output files are not used if the value of the IFLETYP variable in the run management file is set to 1, for these are then supplied to PEST in the run management file itself. Similarly, the model command line as provided in the PEST control file is ignored by Parallel PEST as the model command line is supplied directly to each slave by the user on commencement of the latter's execution (see below). Once the entire PEST input dataset has been prepared, PESTCHEK should be used to ensure that it is all consistent and correct.

Next the Parallel PEST run management file should be prepared according to the specifications provided above.

Before Parallel PEST is started, care should be taken to ensure that the model can run when its execution is initiated in each slave folder. Model input files that are not generated by Parallel PEST (because they contain no adjustable parameters) should be identical across all folders from which the model is run; alternatively each instance of the model should have access to a common set of such files. The model itself will need to be installed on each slave machine.

11.3.2 Starting the Slaves

Go to each of the slaves in turn and open a command line window in its working folder. Start PSLAVE execution in that window by typing the command

```
pslave
```

PSLAVE immediately prompts the user for the command which it must use to run the model. Respond by typing the appropriate command. Remember that, as with normal PEST, the

"model" may be a batch file running a series of executables; alternatively, the model may be a single executable. Provide the model pathname if the model batch file or executable is not cited in the PATH environment variable on the slave's machine, or does not reside in the PSLAVE working folder.

In each slave window, PSLAVE now waits for PEST to commence execution. At this stage, or at any other stage of PSLAVE execution, the user can press the <Ctl-C> keys to terminate its execution if this is desired.

PSLAVE can be started with a special switch that alters its operations on termination of Parallel PEST execution. If a Parallel PEST run comes to a "natural end" (i.e. if its execution is not prematurely terminated by the user and if it does not encounter an error while trying to read model output files), then it signals all slaves that they, too, must shut down when it shuts down. This saves the user the trouble of shutting them down him/herself. However if PSLAVE is started using the command

```
pslave /n
```

it will not cease execution upon termination of PEST execution. This can be useful in circumstances where many Parallel PEST runs must be undertaken in succession (for example in calibration-constrained Monte Carlo analysis). The runs can then be undertaken within a loop written to a batch or script file run from the Parallel PEST master directory. Meanwhile, the slaves need only be started once, before the batch or script file is activated.

11.3.3 Starting PEST

The next step is to start Parallel PEST. Move to the machine on which Parallel PEST resides, open a command line window in the appropriate directory and type

```
ppest case
```

where ***case*** is the filename base of both the PEST control file and the Parallel PEST run management file.

Parallel PEST then commences execution. First it reads the PEST control file and then the run management file. Then it attempts to write to, and read from, each PSLAVE working folder on each machine on which a slave exists in order to verify that it has read/write access to each such folder. It also informs each PSLAVE instance that it has commenced execution; it then waits for a response from each of them. Once it has received all necessary responses it commences the inversion process.

Inversion and predictive analysis algorithms used by Parallel PEST are identical to those used by PEST. The only difference is in model run management. Whenever a model run must be carried out, Parallel PEST selects a slave to carry out this run. If model runs are to be conducted one at a time (as do, for example, the initial model run and the sequence of model runs in which parameter upgrades are tested if the user has decided not to parallelise the lambda testing procedure), Parallel PEST selects the fastest available slave to carry out each run. Initially it knows which slave is fastest from the estimated model run times supplied in the run management file. However once it has completed a few model runs itself, Parallel PEST is able to re-assess relative slave machine execution speeds and will use these upgraded machine speed estimates in allocating model runs to slaves.

The Parallel PEST run manager is "intelligent" to the extent that if a model run is significantly late in completion, Parallel PEST, fearing the worst, allocates that same model run to another slave if the latter is standing idle. Similarly, if a slave has just become free and

Parallel PEST calculates that a model run which is currently being undertaken on a certain slave can be completed on the newly freed slave in a significantly faster time, it reassigns the run to the new slave. As it allocates model runs to different slaves it writes a record of what it does to its "run management record file"; see below.

Parallel PEST execution continues until either the parameter estimation process is complete or the user interrupts it by typing the PPAUSE, PSTOP or PSTOPST command while situated in the PPEST working folder within another command line window; see section 5.4 for further details. In the former case PEST execution can be resumed if the PUNPAUSE command is typed. While its execution is paused, the run record file can be examined by opening it with a text editor.

11.3.4 Re-Starting Parallel PEST

If the RSTFLE variable in the "control data" section of the PEST control file is set to "restart", a terminated Parallel PEST run can be restarted at any time using one of the "/r", "/j" or "/s" switches discussed in section 5.1.5 of this manual. In all of these cases, PSLAVE must be started in the manner described above on each slave machine before restarting Parallel PEST. Note, however, that if PSLAVE is already running on each of these machines, it does not have to be restarted. This is because an already executing PSLAVE can detect the re-commencement of Parallel PEST execution.

If Parallel PEST is restarted without the "/r", "/j" or "/s" switches, it will commence the parameter estimation process from the very beginning. Once again, if PSLAVE is already running on any slave machine (having been initially started for the sake of a previous Parallel PEST run), it need not be restarted. Such a re-commencement of PEST execution "from scratch" will sometimes be warranted after PEST terminates execution with an error message, or if the user terminates PEST execution with the "immediate stop" option; in neither case does PEST signal to its slaves to cease execution, just in case the user wishes to restart PEST immediately after rectifying an error in the PEST control file or in a template or instruction file. Note, however, that if changes are made to the run management file, each of the slaves should be stopped and then re-started.

If Parallel PEST is restarted with the "/r", "/j" or "/s" switches, neither the PEST control file, nor any template or instruction file should have been altered from that supplied to Parallel PEST on its original run. It is important to note, however, that the same does not apply to the run management file. Thus Parallel PEST can re-commence a lengthy execution using more, less, and/or different slaves from those that were used for the initial part of the Parallel PEST run, as long as the new run management file is prepared in the correct fashion and PSLAVE execution is re-commenced on each of the slave machines identified in that file before PEST execution is re-commenced.

11.3.5 Starting Slaves Late

Actually, there is no need for all slaves involved in a Parallel PEST run to be started before Parallel PEST commences execution. Hence slaves can enter the parallelisation process late. In fact, Parallel PEST will happily commence execution even if only one slave is alive. However, all slaves to which Parallel PEST potentially has access must be cited in the run management file, read by Parallel PEST upon commencement of its execution.

As discussed above, when Parallel PEST is started, it first spends a few moments in checking for the presence of slaves cited in the run management file. After this checking period is over, it ceases execution if no slaves have been detected. However if even one slave has been

detected, it commences the parameter estimation process. At any time during this process others of the slaves cited in the run management file can be started. Once a new slave has been started, PEST will soon detect the presence of this slave and will provide it with model runs to carry out.

The fact that some slaves can enter the parameter estimation process later than other slaves adds flexibility to the Parallel PEST process, for it allows computers to be added as they become available.

11.3.6 Losing Slaves

If, during the course of a Parallel PEST run, a slave machine drops out of the network, PEST will continue execution. If communications are lost during the course of a model run, then PSLAVE executing on the lost machine will not be able to inform PEST of the completion of that model run. PEST will soon grow tired of waiting and allocate that run to another slave. It will thus continue execution with less slaves at its disposal.

Even complete network failure may not result in the termination of a Parallel PEST run, for if one slave is running on the same machine as Parallel PEST, Parallel PEST will be able to continue execution using just that single slave.

11.3.7 Re-Starting Slaves

If a slave is shut down, it can be re-started without having to stop and re-start Parallel PEST. A slave may be shut down, for example, if the owner of a computer being used for a Parallel PEST run wishes to re-claim his/her computer for the less important tasks in which his/her computer is normally engaged. In such a case, execution of the slave on that machine may be terminated by pressing the <Ctl-C> keys when focussed on the slave window. You may have to press these keys a few times to terminate execution of both the model and the slave.

Later on, the slave can be re-started. Simply type "PSLAVE" in the appropriate command line window and supply the command to run the model as requested by PSLAVE. After a while the slave should announce that its presence has been detected by Parallel PEST, and you will notice that it has been given a model run to do. Note however that it may not receive this run immediately; if PEST is testing the effects of different Marquardt lambdas it may not require a model run from this slave for a while (especially if partial parallelisation of the Marquardt lambda procedure is not activated).

It is very important to note, however, that there will be occasions when Parallel PEST will simply not detect the presence of the resurrected slave. This will occur if the slave re-commences execution during the same iteration as that in which its execution was previously terminated. There is a good reason for this. Parallel PEST cannot look for lost slaves unless it knows that they are lost. And it doesn't know that a slave is lost unless it is well overdue in returning model results. A slave is only decreed to be "dead" at the end of an iteration. Parallel PEST can then begin searching for signs of its rejuvenation during the next iteration.

11.3.8 Parallelisation of First Model Run

In its normal mode of operation Parallel PEST begins an inversion process by undertaking a single, non-parallelised model run in order to compute the objective function corresponding to initial parameters, and in order to obtain model output reference values for subsequent finite-difference derivatives calculation. Then it undertakes a series of parallel runs in which parameters are incrementally varied in order to fill the Jacobian matrix. However while the

initial, pre-Jacobian run is underway, slaves are standing idle. Depending on the number of available slaves, and on whether computer time is being paid for (for example on the cloud), this can constitute a waste of time and of money.

This problem can be overcome by initiating execution of Parallel PEST using the "/p1" command line switch. The first model run is then undertaken in parallel with model runs required for the filling of the Jacobian matrix. ("p1" stands for "parameter values #1" or "initial parameter values").

Suppose that Parallel PEST is estimating 100 parameters. If its execution is initiated with the "/p1" switch, Parallel PEST will immediately inform the user that it will undertake 101 model runs (more than this if higher order finite-difference derivatives calculation is being employed). It will then report to the screen (and to its run management file) the number of runs that are finished as it receives news of their completion from its slaves. When all 101 runs have been completed it writes to the screen the value of the initial objective function, together with other information that it would normally record once the initial run was completed. It also announces that the Jacobian matrix for iteration 1 has been calculated. Shortly thereafter, it commences computation of parameter upgrades, initiating a sequence of model runs based on different Marquardt lambdas.

If Parallel PEST is interrupted during computation of this initial run package, it can be restarted using the "/s" command line switch. It will then re-commence execution at the same spot at which its previous execution was interrupted. Parallel PEST should not, however, be restarted with both the "/s" and "/p1" command line switches together. Parallel PEST can figure out for itself the contents of a previously interrupted run package; hence the "/p1" switch is not required for restarting of the previous run.

11.3.9 The Parallel PEST Run Management Record File

Section 5.2 discusses the PEST run record file which records the progress of the parameter estimation process. Parallel PEST produces an identical run record file to that of the normal PEST. It also writes a "run management record file". Like the run record file, the run management record file possesses a filename base identical to that of the PEST control file. However PEST endows this file with an extension of "*.rmr"* (for "run management record"). This file echoes information provided in the run management file. Then, as Parallel PEST execution progresses, it records a complete history of communication between the Parallel PEST master and its slaves.

If PEST (or Parallel PEST) execution is re-commenced using a restart switch, the newly re-activated PEST appends information to the run record file created on the previous PEST (or Parallel PEST) run. The same is not true for the run management record file however, for it is overwritten by a newly re-activated Parallel PEST. This is because, as was mentioned above, there is no necessity for Parallel PEST to employ the same slaves when it re-commences execution as those which it employed in its previous life.

11.3.10 The Importance of the WAIT Variable

The role of the WAIT variable was briefly discussed in section 11.2.5. As was outlined in that section, an appropriate value for this variable gives machines across the network time to respond to the information sent to them by other machines. If WAIT is set too small, the potential exists for conflicts to occur, resulting in a message on the Parallel PEST or PSLAVE screen sent by the operating system. In some cases, as mentioned in section 11.2.5, this message demands an answer which, if not provided, can temporarily remove a particular

slave from the Parallel PEST parameter estimation process.

If the message "access denied" appears on the PEST or PSLAVE screen, this is a sure indication that WAIT needs to be set larger. This occurs when PEST or PSLAVE attempts to delete one of the signal files of Table 11.1 after they have responded to the signal. If they do this before the program which wrote the signal file has closed it (as can happen if WAIT is set too small), then the above message appears. This is not serious, however, as no user response is required and both PEST and PSLAVE ensure that a particular signal file has, in fact, been deleted before they continue with their execution.

The more serious case of a model trying to read or write a file that is still opened by PEST results in the operating-system generated message:

```
Sharing violation reading drive C
Abort, Retry, Fail?
```

to which a response is demanded. If this occurs and you are there to respond, simply press the "r" key. If you are not there to respond, the model cannot run; furthermore the affected slave can take no further part in the inversion process until the "r" key is pressed.

Experience will dictate an appropriate setting for WAIT. However it is important to note that a user should err on the side of caution rather than setting WAIT too low. A high setting for WAIT will certainly slow down communications between PEST and its slaves. It will also result in a longer time between the issuing of a PPAUSE or PSTOP command and a response from Parallel PEST. However it will ensure stable Parallel PEST performance across a busy network.

In general, the busier is the network, the higher should WAIT be set. In most cases a value of 1.0 to 2.0 seconds is adequate, even for a relatively busy network; however do not be afraid to set it as high as 10.0 seconds (or even higher) on an extremely busy network. While this could result in elapsed times of as much as 1 minute between the end of one model run and the beginning of another, if this is small in comparison with the model execution time, then it will make little difference to overall Parallel PEST performance. Also be aware that while networks may seem relatively quiet during the day, they may become extremely busy at night when large backing up operations may take place.

11.3.11 If Parallel PEST does not Respond

As was mentioned above, if the PPAUSE, PSTOP or PSTOPST command is issued while Parallel PEST is running, its execution will be interrupted in the usual way. However unlike serial PEST, Parallel PEST does not need to wait until the end of the current model run to respond to these commands; rather there is only a short delay, the length of this delay depending on the setting of the WAIT variable. Hence if WAIT is set extremely high, be prepared for a short wait between the issuing of any of the above commands and a response from Parallel PEST.

There is, however, one particular situation that can result in a large elapsed time between the issuing of any of the above commands and the reception of a response from Parallel PEST. If, when PEST tries to read a model output file, it encounters a problem, it does not immediately terminate execution, reporting the error to the screen. Rather it waits for 30 seconds and then tries to read the file again. If it is still unsuccessful it waits another 30 seconds and tries to read the file yet again. This time, if the error is still present, it terminates execution, reporting the error to the screen in the usual fashion. By trying to read the model output file three times in this way before declaring that the model run was a failure (for example because the

parameter set that it was using was inappropriate in some way, or an instruction file was in error), it removes the possibility that network problems have been the cause of PEST's failure to read the model output file. While it is engaged in this process however, it does not check for the presence of file *pest.stp* (the file written by programs PPAUSE, PSTOP and PSTOPST). Hence, if you type any of these commands while Parallel PEST is thus engaged, you may have to wait some time for PEST to respond. Meanwhile PEST records on the run management record file that there is a

```
problem (either model or communications) in reading results from slave "xxx"
```

11.3.12 An Example

Once PEST has been installed on your machine, a subdirectory of the main PEST directory (i.e. folder) named *ppestex* will contain all the files needed to undertake a Parallel PEST run on a single machine. Before running this example make sure that the PEST folder is cited in the PATH environment variable.

Open a command line window in the *ppestex* folder and create two subfolders of this folder named *test1* and *test2*.

Now open two more command line windows. In one of these windows transfer to subfolder *test1* and type the command

```
pslave
```

When prompted for the command to run the model, type

```
..\a_model
```

Do the same in the other command line window for folder *test2*.

In the first command line window (i.e. the window that is open in the *ppestex* folder) type the command

```
ppest test
```

Parallel PEST should commence execution and, after verifying that it can communicate with each of its slaves, undertake parameter estimation for the *a_model* model.

If you wish, you can also carry out the parameter estimation process using the normal, serial, version of PEST. While situated in the *ppestex* folder type

```
pest test
```

For this particular case serial PEST runs faster than Parallel PEST. This is because the model run time is too small to justify Parallel PEST's run management overheads. Note, however, that Parallel PEST's speed can be increased somewhat by reducing the value of the WAIT variable from that provided in the *test.rmf* run management file.

11.4 BEOPEST

11.4.1 Credits

BEOPEST was introduced to the PEST family of programs by Willem Schreuder of Principia Mathematica. Doug Rumbaugh of Environmental Simulations helped port it to Windows. I am indebted to both of these people.

As a quick reading of the information below rapidly demonstrates, BEOPEST should normally be used instead of Parallel PEST. Reasons for this include the following.

- It is easier to use (no run management file need be prepared).
- Communication between master and slaves is much faster (almost instantaneous).
- Flexibility of machine usage is greater and fewer network permissions are required; basically, if another machine can ping the master's machine, then that machine can run one or more slaves.
- BEOPEST supports the use of multiple command lines (and therefore, through the use of PEST's observation re-referencing functionality, the use of surrogate models). Parallel PEST cannot do this, as the Parallel PEST slave can use only a single command to run the model. Multiple command lines and observation re-referencing are discussed later in this manual.

11.4.2 Some Concepts

In many ways operation of BEOPEST is similar to that of Parallel PEST. The inversion algorithm is identical to that of Parallel PEST (and to the normal PEST). The management of model runs by BEOPEST is identical to that of Parallel PEST, this including the parallelisation of Jacobian runs and the partial parallelisation of runs undertaken for the purpose of testing parameter upgrades calculated using different Marquardt lambdas.

A fundamental difference between BEOPEST and Parallel PEST is that BEOPEST does not use message files for communication between the master and its slaves. Instead TCP/IP is used as the communications protocol. The passing of messages is therefore much faster and latency is reduced. There is no need for a WAIT variable to slow communications so that the operating system can catch up.

Another important difference between BEOPEST and Parallel PEST is that the BEOPEST master does not need permission to write files to a slave's working folder. This is because it never has to write to that folder. Instead, BEOPEST slaves are "smart" in that they write model input files themselves and read model output files themselves using PEST template and instruction files. The master passes the slave (using TCP/IP) the parameter values that it would like it to use on a particular model run. The slave then writes model input files and initiates the model run. When the model run is complete the BEOPEST slave reads model output files using instruction files. Model-generated observations are then passed back to the BEOPEST master using TCP/IP.

An outcome of the fact that model input files are written locally and read locally is that template and instruction files pertinent to the current PEST run must reside on each slave machine; (normally a separate copy of these will reside in each slave folder). A copy of the PEST control file must reside in each slave folder. The BEOPEST slave reads the model command line from the PEST control file resident in its working folder; it is not informed of the model command line by the user. Naturally, all files needed to run the model must also be present in each slave folder. As for Parallel PEST, the working folders used by all slaves must be different; if this is not the case confusion will arise as to which model outputs are linked to which slave.

The simplest setup procedure is for all files required by both PEST and the model to reside in a single folder. The contents of this folder can then be copied to each slave folder prior to commencement of a BEOPEST run. One of these slave folders can also be the master folder.

The BEOPEST slave is actually BEOPEST itself. BEOPEST knows through its command line switches whether it is being run as a master or as a slave. This is further discussed below.

11.4.3 Versions of BEOPEST

In the WINDOWS environment, two versions of the BEOPEST executable program are provided, namely *beopest32.exe* and *beopest64.exe*. In the description that follows, use of the BEOPEST64 executable is assumed.

BEOPEST must always be run using a "/h" switch. The manner of usage of this switch allows BEOPEST to determine whether it is being run as a master or slave.

11.4.4 Running BEOPEST as the Master

To run BEOPEST as the master, use a command such as the following while situated in the master directory (i.e. folder):

```
beopest64 case /h :4004
```

where *case* is the filename base of the PEST control file. The master folder can coincide with a slave folder; this is recommended practice, but does not have to be the case. It must contain at least the PEST control file and files that are cited in the PEST control file. It will be to this folder that the run record file and all other files produced by BEOPEST to record the status and progress of the inversion process are written.

In the above command, "4004" is the port number. This number can be replaced by the number of any unused port. A space must separate "/h" from the colon that precedes the port number; an upper case "H" can be used if desired.

As for Parallel PEST, BEOPEST can be restarted using the "/r", "/j" or "/s" switches; see section 5.1.5 of this manual. For the last of these cases the above command becomes

```
beopest64 case /s /h :4004
```

A similar protocol is followed for the other restart switches. Similarly, BEOPEST can be instructed to read an existing Jacobian matrix file instead of calculating the Jacobian matrix during its first iteration. This is accomplished through use of the "/i" command line option. The above command then becomes

```
beopest64 case /i /h :4004
```

As for the normal version of PEST, BEOPEST will, if started with the "/i" switch, prompt for the name of the JCO file that it must read; see section 5.1.5.

11.4.5 Running BEOPEST as the Slave

In contrast to Parallel PEST, it is recommended that BEOPEST slaves be started after, and not before, execution of the BEOPEST master has been initiated. (Nevertheless the BEOPEST master will normally tolerate the prior commencement of BEOPEST slaves.) Once execution of the master has commenced, slaves can be started any time thereafter, and in any order.

While positioned in a slave working directory, type a command such as the following to run BEOPEST as a slave.

```
beopest64 case /h masterhost:4004
```

where *masterhost* should be replaced by the host name of the machine on which the master resides. Once again, make sure that there is a space between "/h" and the host name. However there should be no space between the host name and the following colon. If you are unsure of the host name of the master, type the command

```
hostname
```

in a command line window of the master machine. Alternatively, instead of the host name, use the IPv4 address of the master. Thus the above command becomes, for example

```
beopest64 case /h 192.168.1.104:4004
```

If you do not know the IP address of the host machine, type the command

```
ipconfig
```

while situated in a command line window on the host machine.

When BEOPEST commences execution as a slave it does some calculations in order to test the speed of its host machine. It writes a "speed index" to the screen and sends this speed index to the master when it announces to the master that it is alive. The master uses this information when allocating runs to slaves. In general faster slaves get preference. The index is updated by the master as model runs are completed.

BEOPEST slaves can be started up at any time during the running of a BEOPEST master. On informing the master that they are alive the master signifies to the screen and to the run record file that it has detected their presence. Model runs are then assigned to them as soon as there are model runs to be carried out.

As for Parallel PEST, slaves can be stopped and started later. If a slave is carrying out a model run when stopped, the BEOPEST master will eventually get tired of waiting for that run to finish and will allocate the run to another slave.

11.4.6 Terminating BEOPEST Execution

Execution of the BEOPEST master can be brought to a halt using the PSTOP and PSTOPST commands in the usual manner. These commands should be issued from a command line window which is open in the folder from which the BEOPEST master is running.

If the PSTOP or PSTOPST command is issued from a command line window open to a slave directory, the slave will cease execution at the end of the current model run, as soon as it has passed the outcomes of that run back to the master.

<Ctl-C> will work in either case as well. Make sure to press these two keys a number of times when stopping the slave so that the model which it is running stops too.

11.4.7 BEOPEST and SVD-Assist

BEOPEST's tasks when undertaking SVD-assisted inversion are much more complicated than when undertaking normal inversion. This is because PEST writes its own *parcalc.tpl* template file at the start of each iteration of the SVD-assisted inversion process; this file contains the information required to calculate base parameter values from current super parameter values. When model input files are written locally by smart slaves rather than by a PEST master which is aware of the folders in which all of its slaves are operating (the latter being the *modus operandi* of Parallel PEST), the PEST master must communicate to each slave the means through which base parameters are re-constructed from super parameters. The BEOPEST master transfers this information to its slaves using the TCP/IP protocol in a manner that is transparent to the user.

While you need have no involvement in this procedure, it is important however that, when preparing for a BEOPEST run, you transfer files from the master folder to slave working folders *after*, and not *before*, SVDAPREP has been run in order to create a super parameter

PEST control file. In particular, this new PEST control file must be transferred to the working folders of all slaves, along with the *picalc.ins*, *picalc.tpl* and *svdabatch.bat* files written by SVDAPREP. The base parameter PEST control file must also be transferred to all slave folders, for the slaves must obtain details of base parameter names, bounds, scales and offsets from this file. Naturally, the name of the super parameter PEST control file written by SVDAPREP must be supplied to both the master and slave versions of BEOPEST through their respective command lines as execution of each of these is initiated.

11.4.8 Parallelisation of Initial Model Run

As described in section 11.3.8, Parallel PEST can be asked to undertake its initial model run as part of the same parallelised run parcel as that used to undertake computation of the initial Jacobian matrix. This avoids the problem of many slaves standing idle waiting for work while the initial model run is being undertaken. This is implemented by starting Parallel PEST with the "/p1" command line switch.

The same applies to BEOPEST; execution of the master must be initiated using the command

```
beopest64 case /p1 /h masterhost:4004
```

As was previously described, if a BEOPEST run is interrupted, it can be restarted using the "/s" command line switch. There is no need to repeat the "/p1" switch to re-commence a BEOPEST run that was previously interrupted during computation of its initial run parcel. BEOPEST will figure out for itself whether or not the "/p1" switch was employed during its previous run from the contents of its restart file.

11.4.9 Multiple Command Lines

If a PEST input file employs multiple model command lines (see the next chapter) BEOPEST will respect this. When sending a parameter set to a slave it will also send the index of the model command that must be used for that particular model run. It is up to the user to ensure that all files necessary for implementing all model commands are accessible from all slave working folders.

(It will be recalled from section 4 of this manual that PEST is instructed to employ multiple command lines by setting the NUMCOM control variable in the "control data" section of the PEST control file to a value which is greater than 1. Parameter-specific command line indices are provided through the DERCOM variable appearing in the "parameter data" section of the PEST control file.)

11.4.10 The Run Management File, PARLAM and RUN_SLOW_FAC

Unlike Parallel PEST, BEOPEST does not need to read a run management file. Recall that this file has the same filename base as the PEST control file, but has an extension of "*.rmf*" As the slaves, and not the master, write model input files and read model output files, the master does not need to know the names and locations of slave working folders. Nor does it need to know in advance of a BEOPEST run how many slaves are at its disposal. It will simply add slaves to its register as they open communications with the BEOPEST master through the TCP/IP protocol, and allocate them runs as long as they are still prepared to implement these runs.

Nevertheless, if a run management file is present within the directory from which the master is launched, the BEOPEST master will read the first two lines of this file. Actually it will only read two variables from this file. The first is PARLAM; the second is the optional

RUN_SLOW_FAC variable. PARLAM is the fourth variable on the second line of the run management file, while RUN_SLOW_FAC can be placed anywhere on this line. Note that BEOPEST will not read the WAIT variable. Hence BEOPEST cannot be instructed to temporarily pause its run management at critical moments so that the operating system can catch up with it; this is unnecessary.

PARLAM

Recall from section 11.2.6 above that PARLAM settings are as follows.

PARLAM setting	PEST action
0	Do not parallelise model runs when testing different parameter upgrades calculated on the basis of different Marquardt lambdas.
1	Parallelise the lambda search procedure. Use all available slaves in this process. Return to serial processing if a parameter hits its bound.
-N	Parallelise the lambda search procedure. Use a maximum of *N* slaves in this process. Return to serial processing if a parameter hits its bound.
-9999	Parallelise the lambda search procedure. Use a maximum of NUMLAM slaves, and undertake only one round of lambda testing.

Table 11.2. PARLAM settings.

As is explained above, a setting of -9999 is the best to use where model run times are long and where a user has access to a moderate to high number of slaves whose run times are similar. In that case it may be wise to set the NUMLAM variable in the "control data" section of the PEST control file to a higher-than-normal value if it would otherwise be smaller than the number of available slaves. (Recall that the NUMLAM variable governs the maximum number of model runs that PEST will commit to the testing of different Marquardt lambdas.) However a value of 15 is normally high enough.

As was stated above, it is important to note that for all PARLAM settings other than -9999, PEST abandons parallelisation of the lambda search procedure if any parameter encounters its bounds. Traditional lambda-based upgrading then becomes a serial procedure as the parameter upgrade direction is re-calculated in a manner that is dependent on the number of parameters that have not yet encountered their bounds. However with PARLAM set to -9999, PEST will, under no circumstances, undertake a second set of model runs during the lambda testing procedure pertaining to any one iteration of the inversion process; nor will it serialize the lambda search procedure. This ensures that no processors are idle during the lambda testing procedure. Where a user has many processors at his/her disposal, some lack of efficiency in conducting the lambda search that is incurred through failure to serialize this search as parameters encounter their bounds, may be more than compensated by efficiencies gained through keeping all processors busy.

If you wish, you can provide BEOPEST with a PARLAM setting of -9999 without having to provide a run management file. If NUMLAM is set to a negative number in the PEST control file, then if PEST is run as Parallel PEST or BEOPEST, the PARLAM variable will automatically be set to -9999. The number of parallel model runs used in the Marquardt lambda testing procedure is then equal to the absolute value of NUMLAM. If a run management file is present, a negative NUMLAM value overrides the PARLAM value setting provided in that file.

RUN_SLOW_FAC

If you wish to provide BEOPEST with a value for RUN_SLOW_FAC then a run management file must be present in the folder in which the BEOPEST master is run. Recall from section 11.2.5 that this variable is optional and can be placed anywhere on the second line of this file. In fact, if supplied to BEOPEST, all other variables can be missing from this line and BEOPEST will not care. Figure 11.7 shows a run management file suitable for the use of BEOPEST in which RUN_SLOW_FAC is assigned a value of 5.0. Because all other variables are missing from this line PARLAM is assigned a default value of 1; alternatively it may have been assigned a value of -9999 through a negative NUMLAM setting in the PEST control file.

```
prf
run_slow_fac = 5.0
```

Figure 11.7 A run management file for the use of BEOPEST in which it is supplied a value for RUN_SLOW_FAC.

BEOPEST will also accept a run management file such as that depicted in figure 11.5 in which other variables are represented on its second line.

11.4.11 Run Management Record File

As for Parallel PEST, the BEOPEST master records all communications between itself and its slaves to a run management record file. The filename base of this file is the same as that of the PEST control file; its extension is "*.rmr*".

11.4.12 Slave Groups

Slave grouping is not available through BEOPEST.

11.4.13 Culling Slaves

As has been previously described in this manual, the PSTOP and PSTOPST commands can be used to halt PEST execution. These utilities write a file named *pest.stp*. This file contains a single integer. PEST monitors its working directory for this file and acts in response to the value of this integer.

Special integer settings are available for BEOPEST. However they are not available through utilities such as PSTOP and PSTOPST. If it is to contain these special settings, file *pest.stp* must be written by the user. These settings are now described.

If *pest.stp* records a value of 10, and if this file is written to a slave folder, then the slave will stop. The slave will also stop if the contents of *pest.stp* are 1 or 2 (values which are written by PSTOP and PSTOPST). However if the slave folder is the same as the master folder, the master will also stop if file *pest.stp* contains a value of 1 or 2. In contrast, a value of 10 will not affect the master, but will precipitate termination of slave execution.

Note that termination of execution of a slave does not affect BEOPEST execution. The BEOPEST master detects the death of a slave; if other slaves are still alive the master then distributes model runs to remaining slaves. If no other slaves are alive the master simply waits for new slaves to appear, or old slaves to reappear before initiating any further model runs.

If *pest.stp* is written to the directory from which the BEOPEST master is operating, and if the

single integer which comprises its contents has a value of *–N*, then the master will cull slaves until there are only *N* of these remaining. It does this by sending commands to culled slaves to cease their execution. Commands to cease execution are distributed to idle slaves immediately. If more slaves must be culled to reach the user-supplied target of *N* remaining slaves, then other slaves are culled after they have finished their respective model runs and have reported their results to the BEOPEST master.

A value of -1000000 must be used for *–N* if it is desired to cull slaves to zero. Even with all of its slaves terminated, the BEOPEST master will not itself cease execution. It will simply wait until a new slave appears, or an old slave re-appears, before distributing any further model runs.

11.4.14 Opening a Port to the Outside World

The following information may prove useful.

Suppose the BEOPEST master is running on one of the machines comprising your office or home network; suppose also that you would like to use a machine that is not part of that network to carry out model runs.

The first thing that you must do is transfer all files required by the model and by BEOPEST to that machine. This can be done in whatever way is most convenient for you.

Suppose now that the BEOPEST master is running on a machine whose local IP address is 10.0.0.4. This address is provided to your machine by your router. It cannot be used by the outside machine, as this address is only recognizable by other machines on your local network.

Suppose that your router is visible to the outside world through an IP address such as 232.213.21.313. Let us assume that you will ask BEOPEST to use port 4004 on its local machine. It is an easy matter to use the port-forwarding functionality of your router to make this port on this machine visible to the outside world as port 4004 associated with the IP address 232.213.21.313. Hence the BEOPEST slave can be run on the outside machine using the command

```
beopest64 case /h 232.213.21.313:4004
```

See your router's manual for further details.

11.4.15 An Example

The example discussed in section 11.3.12 is now repeated using BEOPEST. It is assumed that three command line windows have already been opened, one in the master folder and one in each of the slave folders. (Note that the master and one slave can share the same folder if desired.)

In the master directory type

```
hostname
```

to find out the name of your computer. Suppose this is ***host***. In the master window start the BEOPEST master by typing the command

```
beopest64 test /h :4004
```

Then in each of the slave windows type the command

```
beopest64 test /h host:4004
```

12. Model-Calculated Derivatives

12.1 Contents of this Chapter

The main purpose of the present chapter is to describe PEST's external derivatives functionality. However two not-unrelated topics are also discussed, namely the use of alternative commands to run the model, and PEST-to-model messaging. The issue of alternative model commands is taken up again later in this manual where PEST's observation re-referencing functionality is described.

12.2 Sending a Message to the Model

PEST has the ability to send a small "message" to the model prior to running it; the control variable which activates this functionality is described in section 12.5 below. A modeller can use this to tune some aspect of a model's behaviour to the role that the model plays in different parts of a PEST iteration. The message sent by PEST resides in a file which is always named *pest.mmf*. The contents of a typical message file are shown in figure 12.1.

```
derivative_increment
     -2
      4      20
     hcond1       5.005787             1
     hcond2       9.850230             0
     stor1       -5.660591            -2
     stor2        8.257257        -10000
```

Figure 12.1 A PEST-to-model message file, *pest.mmf*.

The first line of a message file contains a character string which tells the model why PEST is running it. The various strings used by PEST are as follows

forward_model_run

This string informs the model that it is being run either to test a parameter upgrade, as the first model run of the PEST inversion process, or as the final model run undertaken by PEST with optimised parameters.

derivative_increment

"derivative_increment" means that the model is being run as part of the finite-difference derivatives calculation process undertaken by PEST.

external_derivatives

The model is being run in order to write an external derivatives file (see below).

If the character string on the first line of the PEST-to-model message file is "derivative_increment", then the integer on the second line of this file has significance. A value of *n* for this integer indicates that the model run is being undertaken with the value of the *n*'th parameter incremented for the purpose of derivatives calculation by forward differences, or as the first of two runs by which derivatives will be calculated using central differences. A value of *-n* indicates that the *n*'th parameter is decremented in the second of two runs undertaken for the purpose of derivatives calculation by central differences.

The third line of the message file lists the number of parameters (PEST variable NPAR) and number of observations (PEST variable NOBS) involved in the inversion process. Following this are NPAR lines of data with three entries on each line. The first entry on each line is a parameter name; recall that this name can be up to12 characters in length. Then follows the value of the parameter used on the current model run. Following that is an integer code that informs the model of the parameter's status in the inversion process. A value of 0 denotes that the parameter is adjustable and is not logarithmically transformed. A value of 1 indicates that the parameter is adjustable and is logarithmically transformed. A value of *-n* indicates that the parameter is tied to parameter number *n*, while a value of -1000000 indicates that the parameter is fixed.

The PEST-to-model message file is always written to the current working directory; it is written just before each model run is undertaken. However in the case of Parallel PEST and BEOPEST, the message file is written to each slave working directory just before the pertinent model run is initiated by the slave.

12.3 Multiple Command Lines

As will be discussed shortly, when PEST runs a model for the purpose of external derivatives calculation, it can use a different command to that which it uses for ordinary model runs. (The same command can be used for both of these types of model run if desired. In this case it may be necessary for the model to acquaint itself with PEST's expectations for each particular run by reading the PEST-to-model message file.)

PEST also supports the use of different commands to run the model when calculating finite-difference derivatives with respect to different parameters. Recall that when PEST calculates the derivatives of all model outputs with respect to a particular parameter, it runs the model once (more if higher order derivatives are employed) with the value of the parameter incrementally varied. If desired, a different command can be used to run the model when one particular parameter is incrementally varied from that which is used to run the model when other parameters are incrementally-varied. See section 12.5 for implementation details.

Use of a variable command line strategy may promulgate considerable reductions in overall PEST run time in some circumstances. For example, if a composite model is comprised of a sequence of executable programs encapsulated in a batch file, it may not be necessary to run the earlier programs of the sequence when parameters pertaining to the later programs are being incrementally varied for the purpose of derivatives calculation, for outputs of the earlier programs will not then vary between subsequent model runs. Hence the "model" run by PEST when incrementally varying these later parameters may replace the earlier submodel commands with commands by which pertinent output files for these earlier models (stored under separate names) are copied to the model output files expected by PEST (recall that these are deleted by PEST prior to each model run). Caution should be exercised in doing this however, for it must be ensured that the model output files that are copied in this way pertain to un-incremented parameter values for the current iteration. Thus it may be necessary to undertake a full model run for the first of those parameters which affect only the later submodels and, as part of this run, copy model output files from the earlier models to the files which are to be used for temporary storage. This will be done using yet another "model" comprised of a batch or script file which includes the pertinent "copy" commands. However while this strategy will work in a serial run environment, it will not work where Jacobian runs are parallelised. In this case (and indeed in serial cases) use of PEST's model re-referencing functionality provides an easier option for ensuring correct reference values for finite-

difference derivatives calculation; see chapter 14.

The use of multiple command lines can support other labour saving strategies that can reduce the numerical burden of finite-difference derivatives calculation. These include the ability to upgrade initial model states from iteration to iteration, and the use of surrogate models for derivatives calculation. See chapter 14 again.

BEOPEST supports the use of multiple command lines. In contrast, Parallel PEST does not support this aspect of PEST's functionality.

Multiple commands cannot be used if SVD-assist is used as a solution mechanism for the inverse problem.

12.4 Externally-Supplied Derivatives

12.4.1 General

Some models are able to calculate derivatives of their outputs with respect to their parameters themselves. If so, it is often better for PEST to use these derivatives instead of the derivatives that it calculates using finite parameter differences. There are two reasons for this.

1. Code included within the model itself for the purpose of derivatives calculation can often exploit certain aspects of the mathematics underlying the numerical simulation process to calculate derivatives far more quickly than they can be calculated using finite differences.
2. Derivatives calculated directly by the model are often numerically more precise than those calculated by taking differences between model outputs calculated on the basis of incrementally-varied parameter values.

Hence if a model can calculate derivatives itself, PEST should use these derivatives.

PEST provides two options for the reading of model-calculated derivatives. The first is through an "external derivatives file". The second is through an external derivatives file recorded in a format which follows the USGS JUPITER protocol. The native PEST protocol is described first; use of the JUPITER protocol is described later in this chapter.

12.4.2 External Derivatives File

PEST can read derivatives of model-calculated observations with respect to adjustable parameters from a special file written by the model whenever derivatives are requested by PEST. Because this file must satisfy special formatting requirements, it will normally be required that the user add a few lines of code to the model to endow it with the ability to write this file to PEST's specifications.

The "external derivatives file" (as it is called herein) produced by the model must contain a "derivatives matrix" or (if compressed storage is employed in the writing of this file) non-zero elements of a derivatives matrix. A derivatives matrix is different from the Jacobian matrix used by PEST and recorded in its JCO file as the latter matrix takes account of whether parameters are log-transformed, fixed or tied during the inversion process. A model has no knowledge of the transformation status that PEST will award to any of its parameters. Hence it should not take this into account.

The derivatives matrix, like any other matrix, is comprised of rows and columns. Each column contains the derivative of every model outcome for which there is a complementary

observation with respect to a particular parameter. Each row contains the derivatives of a single observation with respect to all parameters.

12.4.3 File Formats

ASCII Uncompressed

The external derivatives file can be an ASCII (i.e. text) file in which numbers are separated by spaces, tabs or a comma (it is read by PEST using free field input). This file must be headed by a line containing two integers. These are the number of columns and rows comprising the following matrix. These numbers must be NPAR and NOBS respectively where NPAR is the number of parameters and NOBS is the number of observations cited in the PEST control file for the current case. The ordering of parameters and observations in the external derivatives file must be the same as that in the PEST control file. Note that a column must be included in the derivatives matrix for *every parameter*, even for those parameters which are tied or fixed (PEST ignores derivatives calculated with respect to fixed parameters). Similarly, there must be a row for *every observation* cited in the PEST control file, even for observations which are assigned a weight of zero.

Where there are many parameters to be estimated, each row representing the derivatives of a particular observation with respect to all parameters can be wrapped onto the next line (or as many lines as you wish). However derivatives for the next observation must begin on a new line.

Figure 12.2 shows an example of an uncompressed ASCII external derivatives file.

```
4  9
5.00000  1707.60  34.4932  42.1234
5.25066  8.79458  93.2321  23.5921
1.04819  1.16448  5.34642  19.3235
1.52323  0.11418  0.59235  75.2354
3.21342  0.48392  9.49293  95.3459
2.49321  5.39230  0.49332  9.22934
19.4492  9.93024  0.49304  5.39234
36.3444  10.4933  0.59439  6.49345
95.4592  86.4234  47.4232  324.434
```

Figure 12.2 An uncompressed ASCII external derivatives file.

ASCII Compressed

PEST provides an option for supplying the external derivatives file in a more compressed format than that illustrated in figure 12.2. This format is depicted in figure 12.3.

```
NPAR NOBS NDIMCOMP
IPAR1 IOBS1 DERIV(IOBS1,IPAR1)
IPAR2 IOBS2 DERIV(IOBS2,IPAR2)
..
ndimcomp times
```

Figure 12.3 Format for a compressed ASCII external derivatives file.

The first line of a compressed external derivatives file should contain three integers, namely NPAR and NOBS (same as for the uncompressed alternative), and another entry named NDIMCOMP. NDIMCOMP must be set to 0, or to a positive number equal to the number of entries to follow. If it is set to 0 (or omitted), the following Jacobian matrix is assumed to adopt the protocol depicted in figure 12.2. If it is set to a positive number, it informs PEST

that NDIMCOMP lines of data follow in this file.

Each of the following lines must contain three entries. The first two entries (IPAR and IOBS) are integers whereas the third ((DERIV(IOBS,IPAR)) is a real number (which can be double precision if desired). IPAR is the parameter number, while IOBS is the observation number; DERIV(IOBS, IPAR) is the element of the derivatives matrix corresponding to observation IOBS and parameter IPAR. Only non-zero elements of the derivatives matrix need to be supplied; missing elements are assumed to be zero.

Compressed storage of an externally-supplied derivatives matrix can result in a dramatic reduction in the size of the external derivatives file where a large number of nonlinear (and hence model-calculated) regularisation constraints are used in the inversion process, for under these circumstances, the derivatives matrix normally has many zero-valued elements.

Binary Compressed and Uncompressed

Binary versions of the compressed and uncompressed external derivatives files described above can be read by PEST. However the following should be noted if supplying these types of file to PEST.

- Optional variables cannot be supplied in binary files. Hence, irrespective of whether a full or compressed matrix is provided, the NDIMCOMP variable must be supplied in the header to a binary external derivatives file. If NDIMCOMP is set to zero, PEST takes this as a signal that the full matrix follows. If it is set to a positive number, this indicates the number of ensuing data lines. If it is supplied as a negative number, PEST ceases execution with an error message.
- Integer values are expected as four byte numbers. However NCOMPDIM is expected to be of eight bytes length if an INTEL compiled version of PEST is employed. (This accommodates very large derivative matrices.)
- Matrix elements are expected as double precision (eight bytes) real numbers.

12.4.4 File Management

If you notify PEST that the model will supply an external derivatives file, then you must also inform PEST of the name of this file, the protocol that it adopts for storage of matrix elements, and of the command which PEST must use to run the model in such a way that it writes this file. (The mechanism for accomplishing all of these is discussed below.) Just before issuing this command (it is issued once for every iteration of the inversion process) PEST first checks to see whether a derivatives file already exists. If such a file does exist, PEST deletes it. Thus if the model fails to run, PEST does not read the old file, mistaking it for the new one; instead it writes an error message to the screen informing you that the derivatives file cannot be found. Alternatively, if PEST issues an error message to the effect that it encountered a premature end to the external derivatives file, this indicates that either the model did not run to completion, or that there is an error in the code added to the model to write this file.

12.4.5 Derivatives Type

As is documented earlier in this manual, some parameters can be log-transformed during the parameter estimation process; PEST then estimates the log (to base 10) of such parameters rather than the parameters themselves. For such parameters, respective elements of the Jacobian matrix used by PEST contain derivatives with respect to the logs of these parameters rather than to the parameters themselves; PEST calculates derivatives with respect

to parameter logs internally from derivatives with respect to native parameters.

When writing the external derivatives file, the model need not concern itself with whether a parameter is log-transformed by PEST or not. The model must simply supply derivatives with respect to untransformed parameters and let PEST take care of the calculations required to convert these derivatives to derivatives with respect to parameter logs.

Similarly, if the SCALE and OFFSET values for a particular parameter differ from 1 and 0 respectively, the model need not concern itself with this. PEST modifies the derivatives cited in the external derivatives file to take account of this.

12.4.6 Tied Parameters

If a parameter is tied to a parent parameter, and derivatives of the former parameter for a particular observation are supplied externally, then derivatives of the tied parameter for that same observation must also be supplied externally. If this does not occur, PEST will cease execution with an appropriate error message.

12.4.7 Combining External and Finite-Difference Derivatives

A complex model often consists of many different parameter types. It may be possible to compute derivatives with respect to some of these parameters inside the model, yet it may be necessary to compute derivatives with respect to others using PEST's traditional method of finite differences. As is discussed below, you can inform PEST of the parameters for which derivatives information is supplied externally, and those for which derivatives must be computed by PEST itself using finite differences.

More complex situations than this can arise. For example, a model may be able to calculate derivatives with respect to a certain parameter for some observations but not for others. In this case, derivatives with respect to the pertinent parameter are actually obtained twice by PEST. First PEST undertakes model runs in the usual manner to calculate derivatives for that parameter using finite differences. Then, after all necessary finite-difference model runs have been undertaken for the purpose of finite-difference derivatives calculation for those parameters which need it, PEST completes the Jacobian calculation process by running the "derivatives model" to calculate external derivatives. As is discussed above, for those observations where derivatives can be calculated by the model, such derivatives are usually more accurate than those calculated by PEST using finite differences; hence they should be used in preference to those calculated by PEST. However in reading the derivatives file it must be ensured that previously-calculated finite-difference derivatives are not overwritten by those elements of the external derivatives matrix that cannot be calculated by the model. To prevent this from happening the respective elements of the external derivatives file should be assigned a value of -1.11e33 by the model. Wherever PEST encounters such a value it does not use it. Rather it uses the derivative value that already exists in its internal derivatives matrix, this having been calculated by finite differences.

In summary, model-calculated derivatives are read by PEST *after* derivatives are calculated by finite differences for those parameters for which the user has requested finite-difference derivatives calculation for at least one observation. Externally supplied derivatives override those already calculated by finite differences except where a value of -1.11e33 is supplied for the derivative value.

12.4.8 Name of the Derivatives File

The user must inform PEST of the name of the external derivatives file in the "derivatives command line" section of the PEST control file for the current case (see below). The external derivatives file can have any legal name except for names of files which are already used by PEST. See Appendix B of this manual for these names.

12.4.9 Predictive Analysis Mode

It is very important to note that if PEST is used in "predictive analysis" mode and at least some derivatives are supplied externally, then the sole member of the observation group "predict" must be the *last observation cited in the PEST control file*. Because the ordering of observations in the PEST control and external derivatives files must be the same, then derivatives for this observation must also comprise the last row of the derivatives matrix contained in the external derivatives file.

12.4.10 Parallel PEST and BEOPEST

PEST cannot receive derivatives through an external file if it is being run as Parallel PEST or BEOPEST.

12.5 PEST Control Variables

12.5.1 General

Variables within the PEST control file that govern PEST's multiple command line, model messaging, and external derivatives functionality are now described.

12.5.2 "Control Data" Section

As is explained in section 4.2 of this manual, the fourth line of the "control data" section of the PEST control file begins with the PEST control variables NTPLFLE, NINSFLE, PRECIS and DPOINT. Three optional variables named NUMCOM, JACFILE and MESSFILE can follow these. If one of these is supplied, then all of them must be supplied. If omitted their default values are 1, 0 and 0.

NUMCOM

NUMCOM is the number of different command lines which can be used to run the model. The actual commands themselves are listed in the "model command line" section of the PEST control file (see below).

Note that when assigning a value to NUMCOM, the command that is used to run the model in order to fill the external derivatives file (if indeed external derivatives are to be used) should not be included in the count. This command is listed in a separate section of the PEST control file to the "model command line" section, as will be discussed shortly.

If there is only one command listed in the "model command line" section of the PEST control file (as will most often be the case), then NUMCOM should be supplied with a value of 1.

JACFILE

If the integer JACFILE variable is provided with a value greater than 0, then a special model run is to be undertaken during each iteration of the inversion process for the purpose of

external derivatives calculation. If external derivatives are not required, then JACFILE should be provided with a value of 0.

A value of 1 for JACFILE signifies that derivatives are to be read from an ASCII file. PEST will know from the header to this file whether the compressed or uncompressed protocol for representation of the derivatives matrix is used. A value of -1 signifies that the external derivatives file is binary.

If JACFILE is provided with a value of 2, this indicates that the external derivatives file adopts the JUPITER storage convention. This convention is discussed later in this chapter.

If JACFILE is provided with a non-zero value and an attempt is made to run Parallel PEST, PEST will cease execution with an appropriate error message.

MESSFILE

Provide this integer variable with a value of 1 if PEST is required to write a PEST-to-model message file prior to each model run. Otherwise provide it with a value of 0.

12.5.3 "Parameter Data" Section

As is described in section 4.9 of this manual, each line of the "parameter data" section of the PEST control file contains values for the variables PARNME, PARTRANS, PARCHGLIM, PARVAL1, PARLBND, PARUBND, PARGP, SCALE, OFFSET and DERCOM (in that order). Only the variable DERCOM is used in implementing PEST's external derivatives functionality.

As will be described below, the various commands which can be used to run the model for purposes other than external derivatives calculation are listed in the "model command line" section of the PEST control file. The value of DERCOM pertaining to each parameter denotes which of these commands should be used to run the model when PEST calculates derivatives with respect to that parameter using finite differences; commands within the "model command line" section are numbered from first to last, beginning at 1.

Alternatively, if the derivatives of all observations with respect to a particular parameter are to be supplied externally, then the DERCOM value for that parameter should be supplied as zero. If this is the case, PEST will not undertake any model runs to calculate derivatives with respect to this parameter using the finite-difference method.

For a particular parameter, derivatives for some observations may be calculated using finite differences while derivatives for others may be supplied externally by the model. In this case a non-zero value should be provided for DERCOM, thus ensuring that PEST calculates derivatives using finite differences for this parameter. If JACFILE (in the "control data" section of the PEST control file) is set to 1, then the model will be called specifically to calculate external derivatives after PEST has calculated derivatives using finite differences for all parameters which are provided with a non-zero DERCOM value. *As is stressed above, to ensure that a finite-difference-calculated derivative is not overwritten when PEST reads the derivatives matrix from the external derivatives file, the model should fill all elements of the derivatives matrix for which it has not actually calculated an external derivative with a value of -1.11E33.*

It is important to note that if JACFILE is provided with a value of 1, then the external derivatives command will be issued, and PEST will read derivatives from the external derivatives file, whether or not any parameter has been assigned a DERCOM value of 0.

Thus if JACFILE is set to 1, PEST can only assume that for at least one parameter with a non-zero DERCOM value, finite-difference-calculated derivatives for some observations are to be supplemented by external derivatives for others. It is again emphasised that when writing code to fill the external derivatives matrix, you should take particular care to provide each element of this matrix with a value of -1.11E33 unless an external derivative is actually calculated for that element. Thus derivatives with respect to parameters for which external derivatives are not required will not be overwritten by spurious values.

12.5.4 "Derivatives Command Line" Section

If a non-zero value is supplied for JACFILE in the "control data" section of the PEST control file, the PEST control file must contain a "derivatives command line" section. This must be situated just above the "model command line" section. Contents of the "derivatives command line" section of the PEST control file are illustrated in figure 12.4, while an example is provided in figure 12.5.

```
* derivatives command line
command to run the model
EXTDERFLE
```

Figure 12.4 Structure of the "derivatives command line" section of a PEST control file.

```
* derivatives command line
model_d.bat
derivs.dat
```

Figure 12.5 An example of the "derivatives command line" section of a PEST control file.

Like all other sections of the PEST control file, the beginning of the "derivatives command line" section must be denoted using a special header line, the first character of which is an asterisk. Following the header line, the next line of this section consists of the command used to run the model when it is required to calculate external derivatives. If appropriate, this can be the same command as that used to run the model for the purpose of testing parameter upgrades or for the calculation of derivatives using finite differences.

The final line of the "derivatives command line" section consists of the name of the file to which the model should write the derivatives matrix, i.e. the name of the external derivatives file.

If JACFILE is set to 0 in the "control data" section of the PEST control file, the "derivatives command line" section can be omitted.

12.5.5 "Model Command Line" Section

Normally the "model command line" section of the PEST control file will contain only a single line, this being comprised of the command that PEST must use to run the model. This is PEST's expectation if the NUMCOM variable is omitted from the PEST control file or if it is set to 1. However if NUMCOM is set to *N*, then there must be *N* model command lines listed in this section of the PEST control file, one under the other. The DERCOM variable in the "parameter data" section of the PEST control file refers to these commands by number when indicating which of these commands is to be used when running the model to calculate derivatives using finite differences with respect to each parameter.

It is important to note that when the model is run in order to test a parameter upgrade, and when the model is run for the first time in the inversion process in order to obtain the

objective function corresponding to the initial parameter set, the first of the listed model commands is used to run the model.

12.6 JUPITER Protocol for External Derivatives

12.6.1 General

PEST has the ability to acquire model-calculated derivatives which are stored using the USGS JUPITER format. If this protocol is adopted for the reading of external derivatives, certain rules must be obeyed. Fortunately, these rules are not very restrictive; if they are transgressed, PEST (and PESTCHEK) will soon remind you of this with an appropriate error message.

12.6.2 JUPITER Protocol for Model-Calculated Derivatives

The following extract from Banta et al (2006) explains the protocol which must be observed for the supply of derivatives to a JUPITER application by a model. Note that a "dependent" is a model output corresponding to an observation, simply referred to as an "observation" in this manual. The extract begins with the following paragraph and extends to the end of this subsection.

A "derivatives interface file" provides a JUPITER application with information needed to obtain model-calculated sensitivities (derivatives of dependents with respect to parameters) from a model output file. All items are read in free format. The format for a derivatives interface file is shown in table 12.1.

Item	Variable name or literal contents	Explanation
0	# Text	Zero or more comment lines allowed only at the top of the file. Comments are identified by # in column 1. Comments also may follow values to be read on each line, other than lines containing names of parameters or dependents.
1	DERFILE	Name of the model-generated file containing derivatives.
2	NSKIP	Number of lines at the top of file DERFILE to skip before reading derivative values.
3	NDEP NPAR	Number of dependents; number of parameters.
4	ORIENTATION	Either “ROW/DEP” or “ROW/PAR”. Enter with or without quotes. See note 1 below.
5	DERFORMAT	FORTRAN format for reading derivatives values, or: “(FREE)”. See note 2 below.
6	“PARAMETERS”	Enter the word “PARAMETERS”, with or without quotation marks. Interpretation is case-insensitive.
7	Parameter names	NPAR parameter names. The names correspond, in order, to the parameters for which model-calculated derivatives are provided in file DERFILE. See note 3 below.
8	“DEPENDENTS”	Enter the word “DEPENDENTS”, with or without quotation marks. Interpretation is case-insensitive.
9	Dependent names	NDEP dependent names. The names correspond, in order, to the dependents for which model-calculated derivatives are provided in file DERFILE. See note 3 below.

Table 12.1. Derivatives interface file format.

Notes on table 1.

1. ORIENTATION: The choice supports reading either an untransposed or a transposed Jacobian matrix from the model-generated file. “ROW/DEP” would indicate that each row in file DERFILE contains derivatives for one dependent. “ROW/PAR” would indicate that each row contains derivatives for one parameter.
2. DERFORMAT: The format string needs to include the parentheses. Single or double quotes may be used to include embedded spaces or commas. Length limit: 200 characters.
3. Parameter names and dependent names. Names are read until NPAR (or NDEP) names are read, the names should be in the order used in the model-generated derivatives file. Multiple names may be listed on each line. The names need to correspond to parameter or dependent names defined elsewhere in the program input.

Figure 12.6 illustrates a JUPITER derivative interface file.

```
# Derivatives Interface File for tc1 model
tc1._su          (DERFILE)
1                (NSKIP)
35  9            (NDEP NPAR)
row/dep          (ORIENTATION)
```

```
'(20x,9f15.0)'  (DERFORMAT)
PARAMETERS
wells_tr  rch_zone_1  rch_zone_2  rivers  ss_1  hk_1  vert_k_cb  ss_2  hk_2
DEPENDENTS
h1.0  h1.1   h1.12  h2.0  h2.1   h2.2  h2.8  h2.12  h3.0  h3.1  h3.12  h4.0
h4.1  h4.12  h5.0   h5.1  h5.12  h6.0  h6.1  h6.12  h7.0  h7.1  h7.12  h8.0
h8.1  h8.12  h9.0   h9.1  h9.12  h0.0  h0.1  h0.12  SS    TR3   TR12
```

Figure 12.6 A JUPITER derivatives interface file.

12.6.3 Using JUPITER Derivatives Protocol with PEST

Instructing PEST to employ JUPITER protocol to acquire model-calculated derivatives is quite straightforward, involving only the following steps.

1. The control variable JACFILE must be set to 2 rather than 1.
2. Each parameter must be supplied with a DERCOM variable as appropriate (same as for traditional PEST external derivatives functionality).
3. A "derivatives command line" section must be present within the PEST control file. The first line in this section must be the command to run the model for the purpose of derivatives calculation.
4. The second (and only other) line in the "derivatives command line" section must be the name of the JUPITER derivatives interface file. (In contrast, for PEST's traditional external derivatives functionality, this is the name of the model-generated file that actually contains the parameter derivatives.
5. The derivatives interface file must be prepared according to the protocol described above.

Thus use of JUPITER external derivatives protocol only requires that the derivatives interface file be prepared, that JACFILE be altered from 1 to 2, and that the EXTDERFLE variable in the "derivatives command line" section of the PEST control file now contain the name of the derivatives interface file, rather than the derivatives file itself.

12.6.4 Special Considerations

If DERCOM for a particular parameter is set to zero in the "parameter data" section of the PEST control file, this informs PEST that no finite-difference derivatives calculation with respect to that parameter is required; that is, the derivatives of all observations with respect to that parameter are supplied by the model. If this is the case, the NDEP variable in the derivatives interface file must be equal to NOBS – the total number of observations. On the other hand, if NDEP is not set to NOBS in a particular derivatives interface file, this indicates that there are some observations for all parameters for which derivatives must be calculated by finite differences, and hence DERCOM must be set to a non-zero value (normally 1) for each parameter.

If NDEP is equal to NOBS, and NPAR in the derivatives interface file is not equal to NPAR in the PEST control file (i.e. the total number of parameters), then parameters are effectively subdivided into two groups – those for which derivatives for all observations are supplied by the model, and those for which derivatives for all observations must be calculated by finite differences. DERCOM for all of the former parameters must be set to zero; for all of the latter parameters it must be set to an appropriate non-zero value. If NDEP is equal to NOBS, and NPAR in the derivatives interface file is equal to NPAR in the PEST control file (i.e. the total number of cited parameters), then all parameters should have a DERCOM value of zero.

Special considerations apply to tied parameters. The tied parameter is not adjusted, but instead maintains the same ratio with the parent parameter as that supplied through initial parameter values; derivatives of parent parameters are adjusted to accommodate the existence of one or more tied parameters.

If a decision is made to tie a parameter to a parent parameter in the PEST control file, no alterations are required to the derivatives interface file. However certain ties are forbidden. In particular, a parameter cited in a derivatives interface file can only be tied to a parameter that is also cited in the derivatives interface file. This rule ensures that there will be no observations for which derivatives with respect to a tied parameter are calculated by the model while derivatives with respect to the parent parameter are calculated by finite differences. The reverse of this rule also applies. That is, a parameter that is not cited in the derivatives interface file cannot be tied to a parameter that is cited within this file. This will rarely be a harsh restriction, for normally it is only parameters that are similar to each other that are tied to each other; derivatives with respect to similar parameters are normally calculated by similar means. If you forget this rule and inadvertently tie parameters together for which derivatives are indeed calculated by different means, PEST or PESTCHEK will detect the error and remind you of this.

When using JUPITER protocol for supplying model-calculated derivatives to PEST when running in "predictive analysis" mode, there is no need to place the sole member of the observation group *predict* last in the list of observations. As in normal PEST operation, this can be placed anywhere within the "observation data" section of the PEST control file.

Due to the fact that parameters and observations for which external derivatives are available are specifically identified in the derivatives interface file, a value of -1.11e33 signifying non-availability of a model-calculated derivative for a certain parameter with respect to a certain observation is not required. Hence this value is treated like any other.

External derivatives can be supplied for a parameter irrespective of whether that parameter is adjustable or fixed. Thus the decision to fix a parameter can be made without reference to the manner in which derivatives are obtained.

12.7 Model-Calculated Derivatives and SVD-Assist

12.7.1 External Derivatives

Model-generated derivatives of base parameters can be employed when PEST is undertaking SVD-assisted parameter estimation, provided super parameters are not supplied externally (i.e. provided the SVDA_EXTSUPER variable in the "svd assist" section of the PEST control file is not set to 1). When external derivatives of base parameters are supplied, PEST re-formulates super parameters during every iteration of the SVD-assisted inversion process; it then calculates derivatives with respect to these super parameters on the basis of the same linear combinations of base parameters as are employed for definition of the super parameters themselves.

When undertaking SVD-assisted inversion, external derivatives functionality is activated in the same way as it is for non-SVD-assisted inversion. That is, the JACFILE variable on the fourth line of the "control data" section of the SVD-assist PEST control file is set to 1. As well as this, the DERCOM value for all super parameters in the "parameter data" section of the PEST control file should be set to zero; thus PEST is prevented from calculating any super parameter derivatives by finite differences.

The following should be noted.

1. As presently programmed, external derivatives can only be read from a file in which the derivatives matrix is stored in ASCII format, and which does not use compressed protocol. Nor can external derivatives be read from a file which employs the JUPITER protocol. Failure to read a JUPITER file is not expected to be a disadvantage for inversion problems that feature a large number of parameters; under these circumstances there is a large computational overhead in reading a JUPITER derivatives file because parameters and observations can, theoretically, be arranged in any order in that file.
2. In the non-SVD-assist context, a value of -1.11e33 in an external derivatives file indicates that the derivative is not supplied in that file; it is then presumed that PEST calculates the derivative using finite parameter differences. This option is not available when external base parameter derivatives are supplied in the SVD-assist setting, because PEST has no way of calculating finite-difference derivatives for base parameters. If such a value is supplied in an external derivatives file, PEST will issue an "out of range" error message.
3. When undertaking SVD-assisted inversion with external derivatives, it is still necessary for a set of original base parameter derivatives to exist in a JCO file prior to the commencement of PEST execution – just as in normal SVD-assist operation. These derivatives are used for initial definition of super parameters, initial computation of super parameter derivatives, and for the setting of internal parameter scaling variables. While this is a little cumbersome it constitutes minimal departure from normal PEST operation.

12.7.2 SVDAPREP

If the JACFILE control variable is set to 1 in a base parameter PEST control file whose name is supplied to SVDAPREP, SVDAPREP will write a PEST control file for SVD-assisted parameter estimation in which external derivatives computation is also activated. Note the following facets of SVDAPREP operation under these conditions.

1. If JACFILE is set to 1 in a base parameter PEST control file, then a "derivatives command line" section must be present within the same PEST control file.
2. Both the model command (in the "model command line" section of the base PEST control file) and the derivatives command (in the "derivatives command line" section of the base PEST control file) must be the names of batch (on a PC) or script (on a UNIX platform) files. In the former case they must possess the extension "*.bat*".
3. In normal SVDAPREP operation the model batch file is supplemented with SVDAPREP-generated commands to run PARCALC and (if there is prior information present in the base PEST control file) PICALC. Commands are also added to delete PARCALC-generated model input files. In doing this, SVDAPREP writes a new model batch file named *svdabatch.bat*. The contents of the derivatives batch file are supplemented in a similar manner; the new derivatives batch file is named *svdabatch_d.bat*.
4. Irrespective of whether or not a "derivatives command line" section is present in the base PEST control file, if JACFILE is set to zero in this file, the DERCOM variable for all super parameters is set to 1 in the SVDAPREP-generated super parameter PEST control file, thereby instructing PEST to compute derivatives for all super

parameters using finite differences. Alternatively, if JACFILE is set to 1 in the base parameter PEST control file, SVDAPREP supplies a DERCOM value of zero to all super parameters, thus requiring that the model supply derivatives for all of them. In fact, as discussed above, the model provides base parameter derivatives and PEST computes super parameter derivatives from these.

5. As noted above, if base parameter derivatives are computed externally, PEST re-defines super parameters during every iteration, and computes super parameter derivatives from base parameter derivatives. Thus PEST effectively sets the SVDA_SUPDERCALC SVD-assist control variable to 1 internally. SVDAPREP does not therefore ask the user whether super parameter derivatives are to be calculated automatically for the first PEST iteration if JACFILE is set to 1 in the PEST control file to which it is directed, for this question is irrelevant under these circumstances.

There is a nuance of SVDAPREP operation of which the user should be aware. If JACFILE is set to 1, then prior information cannot be added to a base parameter PEST control file after the JCO file has been written and before SVDAPREP is run. (This is a relatively unusual situation.) If JACFILE is set to 1, PEST requires that prior information derivatives actually be stored in the JCO file. The same applies if SVDA_SUPDERCALC is set to 1 by the user in the SVD-assist control file. However where super parameter derivatives are computed using finite differences, this is not required.

12.8 An Example

A simple example is presented to demonstrate the use of PEST's external derivatives functionality. Files pertaining to this example can be found in the *edpestex* subfolder of the PEST folder after installation.

File *polymod.f* contains the source code for a simple program which computes the ordinates of a third degree polynomial at a number of different abscissae. That is, it computes the function

$$y = ax^3 + bx^2 + cx + d \qquad (12.8.1)$$

for different values of x. It reads these values of x from a file named *poly_x.in* and writes its computed values of y to a file named *poly_val.out*. "Parameter values", i.e. the values of the polynomial coefficients a, b, c and d, are read from a file named *poly_par.in*.

As well as computing values of y corresponding to different values of x. POLYMOD also computes a "prediction", in this case a function of the parameter values given by the equation

$$p = a + 2b + 3c + 4d \qquad (12.8.2)$$

The "prediction" is written to the end of file *poly_val.out* following the computed polynomial values.

POLYMOD also computes a Jacobian matrix; this is a matrix of the derivative of y with respect to each parameter (i.e. a, b, c and d) at each value of x. This is stored internally in the JACOB array (see POLYMOD source code) and written to a derivatives file in the format expected by PEST. The name of this file is *poly_der.out*.

A template file named *poly_par.tpl* has been prepared to complement the "model input file" *poly_par.in*. This file provides spaces for the four parameters a, b, c and d. An instruction file named *poly_val.ins* reads polynomial values and the prediction value from the model output file *poly_val.out*.

Two PEST control files have been prepared. In one of these (*poly.pst*), PEST is asked to run in "estimation" mode, while in the other (*polyp.pst*) it is asked to run in "predictive analysis" mode. In the former case the model's prediction plays no part in the parameter estimation process as it is assigned a weight of zero. PEST is thus asked to estimate values for the parameters *a*, *b*, *c* and *d* by matching computed polynomial values at different abscissae to the "field data" contained in the PEST control file. Because this "field data" was, in fact, model-generated, PEST is able to achieve a very low objective function.

On the fourth line of the "control data" section of file *poly.pst*, the values of the PEST control variables NUMCOM, JACFILE and MESSFILE are set to 1, 1 and 0 respectively. Thus PEST is asked to look to a derivatives file to obtain its Jacobian matrix; no PEST-to-model message file is requested. As is evident in the "derivatives command line" section of *poly.pst*, the expected name of the derivatives file is *poly_der.out*. Note also from the contents of the "derivatives command line" and "model command line" sections of the PEST control file, that the command used by PEST to run the model for the purpose of derivatives calculation is the same as the command that it uses to run the model in order to simply obtain model outputs.

The final entry on each line of the "parameter data" section of file *poly.pst* is the value of the variable DERCOM. In the present instance DERCOM is zero for all parameters; this indicates that, for each parameter cited in the control file, PEST will obtain derivatives of all model outputs with respect to that parameter from the derivatives file *poly_der.out*; i.e. no supplementary model run is required to calculate some derivatives with respect to this parameter using finite differences.

(Note that an OFFSET value of 1.0 is provided for parameter *d*; this circumvents problems that can sometimes arise in the parameter estimation process when a parameter approaches zero; see the discussion of RELPARMAX and FACPARMAX in section 4.2.)

Check the input dataset contained in file *poly.pst*, and the template and instruction file cited therein, by typing the command

```
pestchek poly
```

at the screen prompt. PESTCHEK should report no errors or inconsistencies - just a warning that the command used to run the model for the purpose of derivatives calculation is the same as that used to run the model for the purpose of obtaining normal model outputs. Then run PEST using the command

```
pest poly
```

PEST should quickly reduce the objective function to a very low value.

Now inspect file *polyp.pst*. While file *polyp.pst* is very similar to *poly.pst*, there are some important differences. Through this file PEST is asked to carry out predictive analysis, minimising the value of the "prediction" while keeping the model "calibrated" to the extent that the objective function (based on the match between model outputs and "field data" cited in the PEST control file), remains at or below a value of 1.0. Starting parameter values are the same as those in *poly.pst*. As these are very different from optimal parameter values, PEST effectively works in "estimation" mode for a while, concentrating on lowering the objective function until it starts to "sniff" the point at which the prediction is minimised while maintaining the objective function below the user-supplied threshold (which is 1.0 in the present case). It then calculates the parameter values corresponding to this point.

Once again, PEST receives derivatives from the model by reading the "derivatives file"

poly_der.out.

Check the PEST input dataset using PESTCHEK and then run PEST to obtain the minimum model prediction that satisfies calibration constraints. This should be about 8.60.

You can repeat these PEST runs with derivatives calculated by PEST using finite differences if you wish. For each of the two PEST control files, do the following.

1. Alter the value of JACFILE (sixth variable on the fifth line) to 0.
2. Alter the value of DERCOM for each parameter (last variable on each line of the "parameter data" section) to 1.

Check your work with PESTCHEK and then run PEST.

13. Pareto Mode

13.1 The Pareto Front

When an objective function possesses two components it may not be possible to minimise both components simultaneously by varying parameters, because minimisation of one may compromise minimisation of the other, and vice versa. Under these circumstances it is possible to define a "tradeoff curve" between the two objective functions by varying parameters appropriately; see figure 13.1.

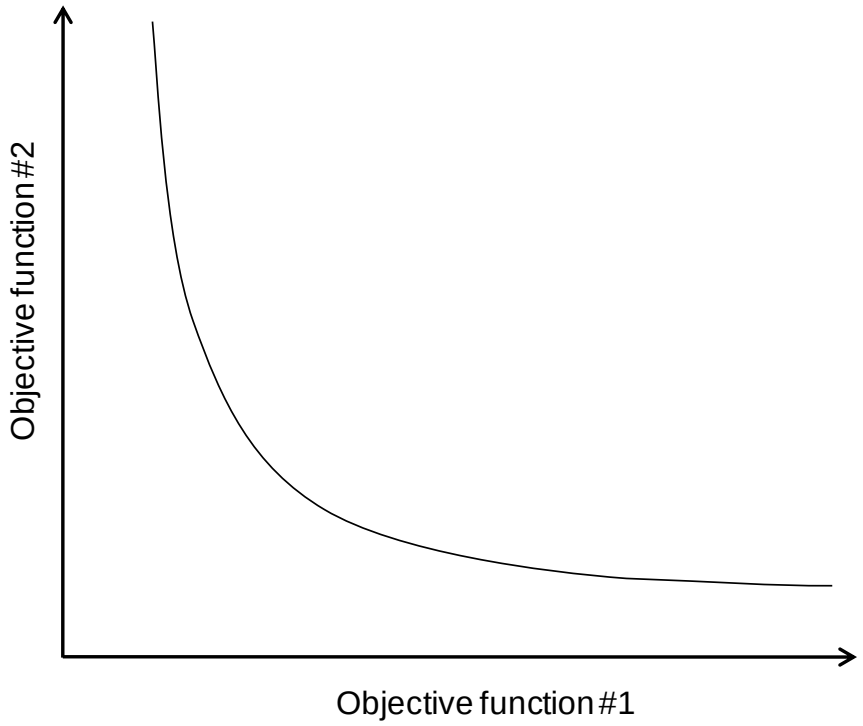


Figure 13.1 A Pareto front defining the tradeoff between two objective functions.

The "Pareto front" is defined as the locus of points in parameter space along which it is not possible to lower one objective function without raising the other. At one end of the Pareto front, one of the two objective functions is minimised, while at the other end of the Pareto front the other objective function is minimised. (Note that a Pareto front can exist between more than two objective function components, in which case its definition is multi-dimensional rather than two-dimensional as in the above case. However only two components are considered in the present discussion, and only two components are supported by PEST's "pareto" mode.)

When run in "pareto" mode, PEST's task is to start at one end of the Pareto front and work its way along the front towards its other end. In most cases of practical interest, there is no need for PEST to actually reach the other end of this front, for much can often be learned through traversing just a part of it.

There are two main uses for PEST's "pareto" mode. One is exploration of predictive uncertainty. The other is exploration of the tradeoff between expert knowledge on the one hand and goodness of fit on the other hand. As such it can implement a "manual" form of

Tikhonov regularisation whereby the user gets to decide him/herself where the loss of expert knowledge in informing parameters is no longer worth the gains in goodness of fit with the calibration dataset.

PEST's Pareto functionality will be described in the context of predictive uncertainty analysis first. Following that, some aspects of its application in the regularised inversion context will be presented. Theory underpinning its use in both of these contexts is provided by Doherty (2015). See section 8.5 of Doherty (2015) in particular. This focusses on model-predictive hypothesis testing, an application to which PEST's Pareto functionality is well suited.

13.2 Exploration of Predictive Credibility

13.2.1 Concepts

The concept of the Pareto front can prove useful in calibration-constrained model predictive uncertainty analysis. In this case one objective function is comprised of members of the calibration dataset. In an ill-posed calibration context this will need to include (extensive) prior information, similar to that used in Tikhonov regularisation. This ensures that parameters remain realistic throughout the ensuing Pareto front traversal process; it also ensures uniqueness of that end of the front. The other objective function component can pertain to one or a number of appropriately-weighted "observations" comprising a prediction of future system behaviour. To the extent that the making of this prediction incurs model-to-measurement misfit, and/or the necessity for parameters to assume unrealistic values, the prediction becomes unlikely, this being measured by the increase in the calibration component of the objective function, the latter rising as the prediction objective function component falls. Through exploration of the tradeoff between the two objective function components, as can be done through definition and traversal of the Pareto front, various predictions can be provided with formal or informal confidence levels.

In exploring the predictive Pareto front, PEST starts at one end of the front (the calibration end) and slowly moves towards the other end. It is assumed that the model has already been calibrated, and that the objective function minimised through this calibration exercise is the same as that defining the calibration end of the Pareto front. (This will be referred to as "the calibration objective function" from now on.) Hence it is assumed that one end of the Pareto front has thereby been located. The other objective function component (henceforth referred to as the "prediction objective function" is then slowly introduced to the inversion process. Weights associated with the observation or observations which contribute to this objective function are zero at first, but then slowly increase at a user-specified rate. As PEST undertakes a sequence of inversion iterations, in each of which the total objective function is minimised (as is its usual behaviour), the prediction objective function thus receives a greater and greater "hearing" in the overall objective function. Hence as the iteration count increases, PEST's location on the Pareto front changes from its original location at the calibration end as it moves towards the prediction end. When the weight associated with the prediction objective function rises to a user-specified level, the journey ends.

It is often unnecessary for this journey to terminate at the prediction end of the Pareto front; however at least part of the front will have been traversed through this process. Hopefully enough of it will have been traversed for predictions that are encountered along the traversed part of the front to be assigned qualitative or quantitative confidence levels, these being based on the degree to which the calibration objective function has been raised in lowering the predictive objective function, and hence in attaining the specified value of the prediction.

PEST-suite utilities which were written to expedite graphical display of the conditions that PEST encounters along the Pareto front can assist in assessing predictive confidence. See also the ASSESSPAR utility; all are documented in part II of this manual.

13.2.2 Implementation

To engage in exploration of the Pareto front PEST must be run in "pareto" mode. For this to occur, the PESTMODE variable in the "control data" section of the PEST control file must be set to "pareto". At the same time, a "pareto" section must be present at the end of the PEST control file. In addition to this, initial parameter values in the PEST control file should correspond to the calibration end of the Pareto front; this is further discussed below.

Figure 13.2 shows specifications of the "pareto" section of the PEST control file. An example of this section is provided in figure 13.3.

```
* pareto
PARETO_OBSGROUP
PARETO_WTFAC_START PARETO_WTFAC_FIN NUM_WTFAC_INC
NUM_ITER_START  NUM_ITER_GEN NUM_ITER_FIN
ALT_TERM
OBS_TERM ABOVE_OR_BELOW OBS_THRESH NUM_ITER_THRESH
NOBS_REPORT
OBS_REPORT_1 OBS_REPORT_2 OBS_REPORT_3
```

Figure 13.2 Specifications of the "pareto" section of the PEST control file.

```
* pareto
prediction
0.0  1.0  20
3  2  1
1
predmaxQ below 50 3
3
predmaxQ simflow2_3 simflow2_4
```

Figure 13.3 An example of the "pareto" section of the PEST control file.

PARETO_OBSGROUP

The second line of the "pareto" section of the PEST control file (the first line contains the "pareto" section header) must contain one entry, this being the name of an observation group cited in the PEST control file. This group must possess at least one observation of non-zero weight. PEST applies a changing multiplier to the weights associated with all observations belonging to this group as a means of exploring the Pareto front. The objective function component associated with this observation group thus becomes one of the two objective function components (the prediction objective function component) in the trade-off which defines the Pareto front. The other objective function component involved in definition of the Pareto front (the calibration component) is the sum of all contributions made to the objective function by all other observation groups featured in the PEST control file.

In many instances of PEST-based Pareto front traversal the PARETO_OBSGROUP observation group will possess only one observation, this being a prediction whose probability of occurrence is being tested through the Pareto front traversal process.

PARETO_WTFAC_START, PARETO_WTFAC_FIN and NUM_WTFAC_INC

The third line of the "pareto" section of the PEST control file must contain two real numbers

followed by an integer, these being respectively the PARETO_WTFAC_START, PARETO_WTFAC_FIN and NUM_WTFAC_INC variables. The first number is the initial weight factor that PEST should apply to members or the observation group PARETO_OBSGROUP (i.e. the predictive observation group). This defines the point at which PEST commences its journey along the Pareto front. In many contexts PARETO_WTFAC_START will be zero, this dictating that the journey commences at one extreme of the front, this being the calibration end of the front.

PARETO_WTFAC_FIN is the final weight factor that PEST must apply to the PARETO_OBSGROUP observation group. Some experimentation may be required to determine a suitable value for this variable. If, however, the user has provided a weight (in the PEST control file) to the single member (or multiple members) of this group which defines the maximum amount that he/she would like the hypothesized prediction to find expression in the total objective function, then a suitable value for PARETO_WTFAC_FIN may be 1.0. However there is no reason why PARETO_WTFAC_FIN cannot be greater than 1.0; indeed an initial Pareto run may determine that it needs to be thus.

Unless PARETO_WTFAC_FIN is provided with a very high value, it cannot be guaranteed that the prediction end of the Pareto front will be encountered in the Pareto journey undertaken by PEST. However, it is normally not necessary that this end of the Pareto front be located; indeed the prediction end of the Pareto front may not even correspond to a unique set of parameters, for there may be more than one way to abandon the constraints of the calibration dataset in achieving a particularly wayward prediction. All that is required in undertaking Pareto-based predictive uncertainty analysis is that the modeller define an appropriate "attractor prediction" or "observed prediction" that is somewhat different from the prediction made by the calibrated model. This prediction must then be assigned a sufficiently high weight for enough of the Pareto front to be explored for likelihoods to be assigned to less extreme predictions.

The final variable on the second line of the "pareto" section of the PEST control file is NUM_WTFAC_INC. This is the number of increments by which PEST should vary the weight factor in raising it from PARETO_WTFAC_START to PARETO_WTFAC_FIN. Thus, for example, if NUM_WTFAC_INC is supplied as 10, PEST will employ 11 (but possibly less - see below) weight factors in at least 10 (but possibly more) successive optimisation iterations in exploring the Pareto front. The first weight factor will be PARETO_WTFAC_START, the second will be PARETO_WEIGHT_START plus a number equal to a tenth of the difference between PARETO_WTFAC_START and PARETO_WTFAC_FIN, etc.

NUM_ITER_START, NUM_ITER_GEN and NUM_ITER_FIN

The fourth line of the "pareto" section of the PEST control file must contain three integers, these being NUM_ITER_START, NUM_ITER_GEN and NUM_ITER_FIN. These variables dictate how many optimisation iterations PEST should devote to minimising the total objective function (i.e. calibration plus prediction objective functions) based on each prediction weight factor that it employs in its journey along the Pareto front. NUM_ITER_START pertains to the first weight factor used (namely PARETO_WTFAC_START); NUM_ITER_START can be zero or greater. NUM_ITER_FIN pertains to the final predictive weight factor used (namely PARETO_WTFAC_FIN); this can also be zero or greater. NUM_ITER_GEN is employed for weight factors in between. This must be one or greater.

ALT_TERM

ALT_TERM, the sole variable on the fifth line of the "pareto" section of the PEST control file, must be 0 or 1. If it is set to 1, then traversal of the Pareto front can terminate prior to using a weight factor of PARETO_WTFAC_FIN. In this case termination of PEST execution will occur if a user-nominated model output (i.e. an "observation" cited in the "observation data" section of the PEST control file) rises above, or falls below, a certain threshold, and stays above or below that threshold over a certain number of iterations. Specifics of this termination criterion must be supplied on the following line; this line must be absent if ALT_TERM is set to zero.

OBS_TERM, ABOVE_OR_BELOW, OBS_THRESH and NUM_ITER_THRESH

The monitored observation is OBS_TERM (first variable on the next line of the "pareto" section of the PEST control file). The next entry on this line (i.e. the ABOVE_OR_BELOW variable) must be supplied as "above" or "below". Then must follow the threshold itself, this being the variable OBS_THRESH. The number of successive iterations for which this threshold must be either exceeded or undercut is supplied as the ensuing NUM_ITER_THRESH variable; this must be 1 or greater. In many instances this observation will be the prediction whose likelihood is being examined through the Pareto front traversal process.

NOBS_REPORT, OBS_REPORT_1, OBS_REPORT_2, etc.

If NOBS_REPORT, on the following line of the "pareto" section of the PEST control file is set to an integer greater than zero, then that number of model outputs (i.e. "observations" cited in the "observation data" of the PEST control file) are monitored and reported to the user as the Pareto front is traversed. If NOBS_REPORT is set to greater than 1, the names of these observations must be listed on the final line of the "pareto" section of the PEST control file, these being the variables OBS_REPORT_1, OBS_REPORT_2 etc. featured on that line.

13.2.3 Preparations for a PEST Pareto Run

It is the user's responsibility to ensure that initial parameter values supplied in the "parameter data" section of the Pareto PEST control file correspond to that point on the Pareto front that is associated with the specified value of PARETO_WTFAC_START. In most cases PARETO_WTFAC_START will be specified as zero; hence initial values must correspond to the calibration end point of the Pareto front. These parameter values can normally be found by undertaking a calibration exercise in which either the PARETO_OBSGROUP observation group is absent from the PEST control file, or in which all members of this group have been assigned weights of zero. (A new PEST control file can be built on the basis of optimised values from a previous PEST run using the PARREP utility discussed in part II of this manual.)

There is no reason why a Pareto PEST control file cannot contain prior information. If so, this will often be the same prior information as that which was employed in a previous Tikhonov calibration exercise. Alternatively, prior information may express equality to previously calibrated parameter values, so that a constraint of minimum departure from these values is maintained as the Pareto front is traversed. As stated above, this prior information will often assume the important role of maintaining parameter reasonableness as high as possible as an hypothesized prediction is tested through traversal of the Pareto front.

Where both prior information and measurements comprising a calibration dataset comprise

constraints associated with the calibration end of the Pareto front, designation of correct relative weighting between both of these may become an important issue. Weights on prior information equations must reflect the innate variability of hydraulic properties, while weights on observations must reflect the degree of measurement/structural noise associated with members of the calibration dataset. Collectively the prior information and observation components of the calibration dataset comprise the total calibration objective function that is traded off against the prediction in undertaking analysis of the uncertainty of this prediction.

Nevertheless, relatively of weighting between prior information on the one hand and field observations on the other will always be problematic. This is because the degree of innate variability that characterises system hydraulic properties on the one hand, and the degree of structural noise with which model-to-measurement misfit is contaminated on the other hand, can only be guessed. Nevertheless, in one form or another, such a decision must be made whenever calibration-constrained predictive uncertainty analysis is undertaken by this or any other means. Some (context-specific) strategies that may be followed in establishing credible relative weighting between prior information equations and observations to which model outputs must be matched include the following.

1. Work in terms of observation and parameter *differences* rather than in terms of *absolutes* when traversing the Pareto front. That is, let the parameters that are adjustable through the Pareto front exploration procedure be departures of parameters from their calibrated values, and let "observations" to which model outputs are matched be model outputs produced on the basis of the parameter set which calibrates the model. Optionally, project prior information onto the calibration null space. Setup for this option can be facilitated through use of the PARREP, OBSPREP and REGPRED utilities described in part II of this manual. See Doherty (2015) for the philosophical basis for such an approach.
2. When using PEST in a Tikhonov-based calibration exercise which precedes Pareto-based predictive uncertainty analysis, do not run PEST in "regularisation" mode. Instead use fixed weights for Tikhonov prior information and observations, these weights reflecting innate hydraulic property variability and measurement/structural noise as discussed above.
3. In the Pareto predictive analysis process, use prior information weights that were calculated by PEST during a previous Tikhonov calibration exercise. Final weights obtained through the latter process can be obtained from the residuals file written by PEST upon conclusion of the regularised inversion process. (It may be important to set the "regcontinue" variable to "continue" during this calibration exercise).
4. Adjust regularisation weights so that the contribution to the calibration objective function made by observations and prior information equations are about the same at the commencement of the Pareto exploration process.

13.2.4 PEST Output Files

Some of the output files which PEST writes when run in other modes are not written when PEST runs in "pareto" mode; this is further discussed below. However two output files are written only when PEST runs in "pareto" mode. These are the "Pareto objective function file", and the "Pareto parameter data file". The first of these is named *case.pod* whereas the second is named *case.ppd*, where *case* is the filename base of the PEST control file. The first is a text file, and can be inspected at any time during the course of the Pareto front traversal process. The second is a binary file; utilities (see below) are available for extracting some or all of the contents of this file and rewriting them in ASCII format.

The Pareto objective function file is easily imported into a plotting package, or into a spreadsheet, for plotting of the Pareto front (or of other information pertaining to this front). Data within this file is arranged in columns; a header is provided for each column to identify its contents. The file is updated at the end of every iteration (including the "zeroth" iteration, this notionally comprising the initial model run based on parameter values supplied in the "parameter data" section of the PEST control file).

The Pareto objective function file contains the contributions to the total objective function made by every observation group cited in the PEST control file, including that whose weight is varied as a means of Pareto front traversal. It is important to note however, that the weight factor applied to this observation group in computing its contribution to the total objective function is 1.0. Hence weights applied to members of this group are full weights as supplied in the PEST control file. By plotting values in this column against the sum of values in other objective function component columns, a picture of at least part of the Pareto front can be obtained (see figure 13.1).

In addition to objective function component columns, the Pareto objective function file contains a further NOBS_REPORT columns. These columns contain the values of those model outputs specified in the last line of the "Pareto" section of the PEST control file. If one of these is the sole member of the observation group whose weight is varied in the Pareto exploration process (this normally being the prediction whose uncertainty is explored), then a plot of this against the sum of all other objective function components provides a picture of how the value of the prediction must be traded off against calibration misfit. In some circumstances it may be possible to use the rise in calibration objective function required to incur specific values of the prediction to calculate predictive confidence intervals using equations 8.3.13, 8.3.14, 8.4.2 and 8.4.3 of Doherty (2015).

The Pareto parameter data file contains the same data as does the Pareto objective function file. However in addition to this it contains parameter values associated with each of the listed objective function values. This file can be translated to ASCII format using the PPD2ASC utility discussed below; note, however, that the width of this ASCII file will be large when many parameters are cited in the PEST control file. Alternatively, parameters listed in an individual record of the Pareto parameter data file can be extracted from that file using the PPD2PAR utility (also discussed below); extracted parameters are written in parameter value file format (see section 5.3.2) by PPD2PAR. From there they can be inserted into a new PEST control file using the PARREP utility; a single model run can then be undertaken on the basis of these parameters by running PEST with NOPTMAX set to zero in this new PEST control file.

In normal operation PEST writes a parameter value file at the end of every iteration of the parameter estimation process. This contains the "best" parameters achieved to date during the current PEST run. The meaning of "best" depends on the mode in which PEST is run. However when run in "pareto" mode, the concept of "best" does not apply, for the process of optimisation is replaced by one of Pareto front traversal. In this case PEST writes a new and distinct parameter value file at the end of each iteration. These files are named *case.par.0, case.par.1, case.par.2* etc., where *case* is the filename base of the PEST control file. The numeric extension associated with the name of each file pertains to the iteration number. An extension of "0" pertains to the end of the zeroth iteration, this being comprised of the initial model run undertaken on the basis of initial parameter values. Parameter values recorded in these files are the same as those recorded in binary form in the Pareto parameter data file (and in ASCII-translated equivalent files written by PPD2ASC).

13.3 Highly Parameterized Inversion as a Trade-off

13.3.1 Concepts

As was described in chapter 9 of this text, when PEST is run in "regularisation" mode, two mega-components comprise the total objective function. One component is named the "measurement objective function" while the other component is named the "regularisation objective function". PEST adjusts weights on observations (including prior information equations) comprising the regularisation objective function such that the measurement objective function approaches (if possible) a user-supplied target value. If observations and prior information equations that collectively define regularisation constraints are pervasive (and adequately reflect knowledge of site conditions), and if the target measurement objective function is set at a value that reflects the level of noise associated with the measurement dataset, the calibrated parameter field will deviate from a presumably maximum pre-calibration likelihood condition described by regularisation constraints only to the minimum amount required to achieve the desired level of fit.

In practice, selection of an appropriate value for the target measurement objective function is a difficult undertaking, for the amount of noise associated with the measurement dataset is unknown. Furthermore, much of this "measurement noise" is actually structural noise (i.e. model-to-measurement misfit arising from model imperfections) which, as is the nature of structural noise, possesses an unknown (though probably singular) covariance matrix. Overfitting is normally deemed to have occurred when the cost of achieving this fit is the introduction of too much heterogeneity to the calibrated parameter field. If this occurs (as it often does), the regularised inversion process is then repeated with the target measurement objective function set higher, for it can be concluded from the previous calibration exercise that there is more "noise" in the measurement dataset than was previously supposed. Eventually a suitable compromise is found between goodness of fit on the one hand, and acceptability of the calibrated parameter field on the other hand. The model is then considered to be "calibrated".

From the above analysis it is obvious that there is an interplay between goodness of fit and calibrated parameter field reasonableness. In practice, a suitable level of model-to-measurement fit is not known in advance of the calibration process because the level of structural noise associated with model outputs corresponding to field measurements is also unknown in advance. In fact the level of structural noise can only be assessed through the "damage" done to parameter fields in achieving various levels of fit. This "damage" can, in turn, only be assessed through the calibration process. Hence part of the role of the calibration process is to seek a suitable compromise between model-to-measurement fit on the one hand and parameter field acceptability on the other hand. In doing this, the calibration process informs the user of at least some of the stochastic characteristics of structural noise. The source of this information is ultimately the user's concept of the level of heterogeneity that realistically prevails in system properties that parameters purport to represent in the model that is currently being calibrated.

13.3.2 Implementation

Pareto concepts can be used to assist in this necessarily subjective process of reconciling goodness of fit with parameter field heterogeneity as it is undertaken through the model history-matching process. In implementing this methodology the modeller begins with a parameter field for which the regularisation objective function is zero. (Note however that

this objective function component is not referred to as such when PEST runs in "pareto" mode; the "expert knowledge" component of the objective function may be a better description of it in this context.) Such a parameter field is normally easy to obtain. In many cases it will be (piecewise) uniform; in other cases, parameter values will simply be their preferred values as suggested by site characterization studies. However you should endeavour to supply weights to prior information equations and regularisation observations that reflect the expected level of hydraulic property variability that prevails at a particular study site. In particular, weights should be roughly equal to the inverse of the standard deviation of expected parameter variability. Alternatively, observations and prior information equations which provide the "soft knowledge" that often encapsulate expert knowledge constraints can be provided with covariance matrices based on one or a number of variograms that are presumed to characterize parameter field variability (see, for example, the PPCOV and PPCOV_SVA programs from the PEST Groundwater Utilities suite, as well as the PLPROC program). Regardless of how weighting is assigned, you should pay particular attention to establishing correct relativity between weights applied to parameters of different types.

Once weights have been assigned, the expert knowledge component of the calibration dataset, taken on its own, should provide a pre-calibration description of parameter site variability as far as this is known by the user – this description including preferred parameter values (and/or parameter relationships), together with a stochastic characterization of the variability of these values/relationships.

If a weight multiplier of zero is applied to all observations comprising the measurement objective function (it is not referred to as such when PEST runs in "pareto" mode), then not only is the expert knowledge objective function equal to zero, for so too is the measurement objective function. Use of such a weight multiplier is most easily attained by running PEST in "pareto" mode, assigning all observations to a single observation group, citing this group as the value of the PARETO_OBSGROUP Pareto control variable, and setting PARETO_WTFAC_START to zero.

As for observations and prior information equations that express expert knowledge, care should be taken to ensure correct relativity of weighting between different measurement types when assigning them to this over-arching observation group. In some cases weighting should be inversely proportional to expected measurement credibility. In other cases, structural aspects of model-to-measurement misfit are best handled by appropriate processing of model outputs and corresponding observations, assigning processed outputs to different groups (before combining them into a single group as discussed above), and ensuring that each such group contributes roughly the same to the starting objective function (using, for example, the PWTADJ1 utility described in part II of this manual). Also, if possible, measurement weighting should be such that the yet-to-be-determined optimised measurement objective function is of the same order of magnitude as the optimised expert knowledge objective function when the weight factor for the former is about 1.0. This increases the chances that a single traversal of the Pareto front with PARETO_WTFAC_FIN set to between 1.0 and 2.0 will locate a point in parameter space that the user considers to be optimal.

PEST can now be run in "pareto" mode; as it does so, measurements comprising the observation dataset are introduced slowly to the overall objective function as PEST increases the weight multiplier that is applied to this overall group. The PARETO_WTFAC_START control variable should be set above zero; better still, it could be set to zero and the NUM_ITER_START variable set to zero. Either of these measures ensures that PEST's first iteration is not devoted to lowering an initial objective function that is already zero (which, of

course, is a futile undertaking); the latter option however adds the zero objective function to the *case*.*pod* file which can be useful for plotting the outcomes of PEST's trajectory of the Pareto front. Meanwhile, PARETO_WTFAC_FIN should be set at a value that hopefully achieves slightly too good a level of model-to-measurement fit. A setting of between 1.0 and 2.0 will presumably achieve this if the above weighting suggestions are followed. Traversal of that portion of the Pareto front so defined will then provide parameter sets from which you can choose one that you consider to provide optimal tradeoff between parameter field credibility on the one hand and goodness of model-to-measurement fit on the other hand.

13.3.3 Operational Details

The methodology just described can be thought of as a form of computer-assisted manual regularisation. Though computationally costly, it allows the user to establish for him/herself the point at which model-to-measurement fit is best traded off against respect for expert knowledge. The subjective nature of this decision cannot really be avoided. Using PEST in "pareto" mode enhances your ability to exercise your subjectivity to best advantage in making this decision.

To employ Pareto analysis as a means of highly parameterized inversion the following steps should be taken. (Some of these have already been discussed but will be set out again in order to provide a summary of the methodology.)

1. All observations comprising the measurement dataset should be assigned to a single observation group. To the best of your ability, ensure that relativity of weighting between components of this overall group is correct. In some cases this can be achieved by equalizing contributions to the total measurement objective function by different observation groups before combining them into a single group. Also, weights assigned to members of this group should be such that, as far as you can judge, the contribution to the overall objective function made by observations and prior information equations which express expert knowledge on the one hand, and by field measurements comprising the calibration dataset on the other hand, will be about equal at that point on the Pareto front that is considered to be optimal.
2. Parameters should be given initial values which result in a value of zero for the expert knowledge component of the objective function. Meanwhile, weights assigned to observations and prior information equations which contribute to this expert-knowledge objective function should be the inverse of the expected standard deviations of variability of the parameters which they site. Covariance matrices, perhaps based on variograms, can be used to characterize spatial parameter variability where appropriate.
3. PESTMODE should be set to "pareto".
4. A "pareto" section should be added to the PEST control file. The name of the observation group pertaining to the calibration dataset should be assigned to the PARETO_OBSGROUP variable. With the weighting strategy suggested above, settings for other Pareto control variables should be as follows.
    a. 0.0 for PARETO_WTFAC_START
    b. 1.5 to 2.0 for PARETO_WTFAC_FIN
    c. 15 to 20 for NUM_WTFAC_INC
    d. 0 for NUM_ITER_START
    e. 2 for NUM_ITER_GEN
    f. 2 for NUM_ITER_FIN
5. Run PEST.

It is important to note that observations and prior information equations that comprise regularisation constraints do not need to belong to an observation group whose name begins with "regul", as must be the case when PEST is run in "regularisation" mode. However, it is probably a good idea to maintain this protocol for ease of identification of these entities. Also, PEST does not refer to a "measurement objective function" and a "regularisation objective function" when run in "pareto" mode. However you can make this association yourself when allocating names to observation groups used in the Pareto process.

PEST's behavior when run in "pareto" mode, as well as the screen outputs that it produces and the files which it writes, were described above. At any stage of a PEST run, a picture of the Pareto front (as traversed so far) can be obtained by plotting data contained in file *case.pod*; parameters corresponding to any point along the Pareto front are available in file *case.ppd* as well as in file *case.par.N*, where *N* signifies the iteration number and *case* signifies the filename base of the PEST control file.

13.4 Some Further Implementation Details

13.4.1 Parallelisation and Pareto

Both Parallel PEST and BEOPEST can operate in "pareto" mode.

13.4.2 Restarting a Pareto Run.

Normal PEST re-start functionality is available when PEST runs in "pareto" mode.

There will be many occasions however when a PEST run is finished and a user, on inspection of the outcomes of this run, would like to traverse more of the Pareto front than was traversed during the just-completed run. This can be achieved as follows.

1. Using the PARREP utility, create a new PEST control file containing the last set of parameters that PEST calculated on its previous run. These will correspond to the end of the Pareto front segment as traversed by PEST on its previous run. (Be sure to give the new PEST control file a different name to that of the old PEST control file so that the *case.pod*, *case.ppd* and *case.par* files from the previous run are not overwritten during the new run.)
2. Set PARETO_WTFAC_START to the previous value of PARETO_WTFAC_FIN, and PARETO_WTFAC_FIN to a suitable value. NUM_ITER_START can normally be set to zero to avoid repetition of the last iteration of the previous Pareto process.
3. Run PEST. The information in the *case.pod*, *case.ppd* and *case.par* files produced on this new run then supplements information contained in files of similar name produced on the previous run.

13.4.3 Special Considerations for SVD-Assist

If you wish to use SVD-assist for exploration of the Pareto front, this is easily accomplished. As is normal protocol for SVD-assisted inversion, the following steps should be taken.

1. Set up a Pareto run in the manner described above.
2. Set NOPTMAX to -1 or -2 in the PEST control file.
3. Run PEST to produce a JCO file.
4. Run SVDAPREP in order to produce a PEST control file for SVD-assisted Pareto front exploration.

When PEST is run in "pareto" mode when undertaking SVD-assisted inversion, the SVDA_MULBPA variable is automatically set to 1. Thus multiple *case.bpa.N* files are saved – one at the end of each iteration. As usual, these are named after the base parameter PEST control file; they contain base parameter values corresponding to points along the Pareto front as defined in file *case-svda.pod*; note how the filename base of the "*.pod*" file corresponds to that of the super parameter PEST control file, this being the PEST control file that is actually used for Pareto front traversal. Under these circumstances, PEST does not leave a succession of "*.par*" files, as the super parameter values that would be recorded in files *case-svda.par.N* are of no real use to the modeller.

When undertaken SVD-assisted traversal of the Pareto front, parameter values recorded in the "*.ppd*" file (i.e. in file *case-svda.ppd*) are actually base parameter values. This makes it easy for the user to link points on the Pareto front (as defined by objective functions listed in file *case-svda.pod*) to parameter values as actually used by the model.

An extra setting is available when implementing SVD-assisted exploration of the Pareto front. This is supplied through the SVDA_PAR_EXCL control variable which resides in the "svd assist" section of the PEST control file immediately following the SVDA_SUPDERCALC variable. SVDA_PAR_EXCL must be set to 0, 1 or -1. If it is set to 1, super parameters are defined using only sensitivities of model outputs that comprise the observation group whose weight factor is adjusted during the Pareto front exploration process; this is the observation group whose name is provided as the PARETO_OBSGROUP variable in the "pareto" section of the PEST control file. In the regularisation context, members of this observation group are the measurements that are fitted during the calibration process. Prior information equations and observations which express expert knowledge are thus ignored in defining super parameters. Limited experience to date suggests that SVDA_PAR_EXCL is best set to 1 when using the Pareto process as a manual regularization device.

If SVDA_PAR_EXCL is set to 0, super parameter definition is based on all observations and prior information equations cited in the base PEST control file (using the weights provided in that file).

If SVDA_PAR_EXCL is set to -1, then super parameters are computed on the basis of all observation groups in the PEST control file other than that named as the Pareto-adjustable group in the super parameter PEST control file. This is useful if traversal of the Pareto front takes place in the opposite direction – that is, from a fit which is initially too good (based on parameters which are too heterogeneous) to a fit which is not as good but which is based on more acceptable parameters (see below).

When SVDAPREP is used in order to build a super parameter PEST control file, it checks the setting of the PESTMODE variable in the base parameter PEST control file. If this is set to "pareto" SVDAPREP asks the user for an appropriate setting for SVDA_PAR_EXCL. Its prompt (SVDAPREP's final prompt) is:

```
Provide setting for SVDA_PAR_EXCL [0, 1, or -1] (<Enter> if 0):
```

The default is 0 (for this is the best setting to use when using the Pareto concept to explore predictive uncertainty). Respond with "1" to set SVDA_PAR_EXCL to 1 in the super PEST control file written by SVDAPREP, "0" to set SVDA_PAR_EXCL to 0 in this file, or "-1" to set it to -1.

13.4.4 Going the Other Way

Suppose that you have just calibrated a model and that the calibrated parameter field contains too much variability. Suppose that you would like to remove some of this variability by slowly introducing stronger and stronger enforcement of regularisation constraints, thereby maintaining as good a fit with measurements as possible while removing some of the more unsightly aspects of the calibrated parameter field. This is easily achieved by assigning all expert knowledge prior information equations and expert knowledge observations to a single observation group, and assigning the name of that group to the PARETO_OBSGROUP variable in the "pareto" section of the PEST control file. Initial parameter values should be those that provided "too good a fit" with the data. PEST can then reduce variability of this parameter field by moving towards the end-point of the Pareto front where expert knowledge observations dominate measurement observations.

In doing this, it may be helpful to remember that if prior information is added to a PEST control file after a JCO file has been computed for that file, the JCO2JCO utility will automatically include the sensitivities for the additional prior information in the new JCO file which it constructs for the new PEST control file.

If using PEST's SVD-assist functionality to traverse the Pareto front in the reverse direction, consider setting SVDA_PAR_EXCL to -1 in the super parameter PEST control file.

13.4.5 Some Differences and Similarities with Other PEST Modes

When PEST runs in "pareto" mode, it informs the user through its screen output, and through its run record file, of the current value of the weight factor applied to the user-nominated PARETO_OBSGROUP observation group. This is done before the initial model run, and at the beginning of each PEST iteration. Objective function values written to the screen and recorded on the run record file for this observation group are calculated on the basis of this current weight factor (in contrast to objective function component values for this group recorded in the Pareto objective function and parameter data files which use full weighting as supplied in the PEST control file). This provides consistency with other aspects of iteration-specific PEST output, in particular the efficacy of different Marquardt lambdas in lowering of the objective function. Hence information written to the screen and recorded on the run record file is indicative of local optimisation performance using the current PARETO_OBSGROUP weight factor, but cannot be used for definition of the Pareto front.

Differences between normal PEST operation and its operation when run in "pareto" mode all arise from the fact that there is no actual "solution" to a Pareto front traversal process. Hence, as discussed above, multiple, iteration-specific, parameter value files are written in place of a single parameter value file containing "optimised" parameters. Another difference is that the residuals file (i.e. file *case.res*), and (if pertinent) the rotated residuals file (i.e. file *case.rsr*), are not saved at the end of a PEST run. Nor is the intermediate residuals file (i.e. file *case.rei*) saved at the end of every iteration as a matter of course. However if the REISAVEITN control variable is set to "reisaveitn", a suite of iteration-specific interim residual files are saved in the usual manner. The Jacobian matrix file (i.e. JCO file) is saved only at the end of the first iteration. However iteration-specific JCO files are saved in the usual manner if the JCOSAVEITN control variable is set to "jcosaveitn".

Upon completion of execution, PEST does not run the model one last time using optimised parameter values (for there is no definition of "optimised" in the Pareto context). No resolution data file is saved, regardless of the well- or ill-posedness of the inverse problem

with which PEST is faced during all iterations of the Pareto front traversal process, and of the solution mechanism chosen to solve the possibly ill-posed series of problems that it encounters. When run in "pareto" mode, PEST does not complete its run record file with a listing of optimised parameter values and a listing of parameter and observation statistics. Instead it simply announces that the Pareto front exploration process has reached completion.

With the exception of the NOPTMAX variable, termination criteria provided in the "control data" section of the PEST control file are ignored when PEST is run in "pareto" mode, for termination of traversal of the Pareto front is governed by variables residing in the "pareto" section of the PEST control file. However if NOPTMAX is set to zero, then PEST will conduct only one model run before terminating execution; if it is set to -1 or -2 PEST will undertake enough model runs to calculate the Jacobian matrix, and then cease execution. In fact if NOPTMAX is provided with any of these values PEST will change its mode of operation to "estimation" so that its normal suite of outputs is recorded on its normal suite of output files before termination of execution.

If the FORCEN variable in the "parameter groups" section of the PEST control file is set to "switch" or "switch_5", PEST will cease execution with an error message. For the sake of consistency, the means by which finite-difference derivatives are calculated should not change as the Pareto front is traversed.

When run in "pareto" mode, any of the usual methodologies offered by PEST for solution of the inverse problem can be selected. Thus, for example, a Pareto PEST control file can include a "singular value decomposition" or "LSQR" section; it can even specify that automatic user intervention enhance optimisation performance. These mechanisms (especially singular value decomposition and LSQR) may indeed be necessary in ill-posed parameter estimation contexts. In fact, as was stated earlier in this manual, use of either singular value decomposition or LSQR is recommended practice.

13.5 Pareto Support Utilities

13.5.1 The PPD2ASC Utility

The PPD2ASC utility reads data that is stored by PEST in the binary Pareto parameter data file (i.e. file *case.ppd*). This file contains the following information:

1. the contribution made to the objective function by each observation group, both at the commencement of the Pareto front traversal process, and at the end of every iteration that PEST undertakes in traversing the Pareto front (weights applied to the adjustable group are full weights as in the objective function file);
2. the values of user-specified model outputs corresponding to these objective functions;
3. parameter values corresponding to the above.

The first two of the above items are also recorded by PEST in ASCII format in its Pareto objective function file (i.e. file *case.pod*); however storage of parameters is confined to the Pareto parameter data file. Binary storage is employed as this file can become unwieldy in highly parameterized contexts where parameters may number in the hundreds, or even in the thousands.

PPD2ASC stores all of the data recorded in the Pareto parameter data file in tabular ASCII format. It is run using the command

```
ppd2asc ppdfile textfile
```

where

ppdfile is the name of a PEST-generated Pareto parameter data file, and

textfile is the name of the ASCII file to which data housed in this file is transferred.

The text file written by PPD2ASC is readily imported into a spreadsheet for analysis or plotting.

13.5.2 The PPD2PAR Utility

Like PPD2ASC, PPD2PAR reads a binary Pareto parameter data file. It extracts a single parameter set from this file, recording this parameter set in parameter value file format; see section 5.3.2 for specifications of this type of file. Once in this format, parameter values are available for use by a number of other PEST utilities. For example the PARREP utility can be used to write a PEST control file in which these parameter values are featured as initial values. If NOPTMAX is set to zero in this new PEST control file, PEST will run the model once, record objective function components, and then cease execution. Meanwhile model input files will contain these same parameter values, while model outputs will be calculated on the basis of these parameter values.

PPD2PAR is run using the command

```
PPD2PAR ppdfile parfile N
```

where

ppdfile is the name of a PEST-generated Pareto parameter data file,

parfile is the name of the parameter value file whose task it is for PPD2PAR to write, and

N is the iteration number pertaining to the extracted parameter set.

Parameter set *N* pertains to the end of the *N*'th iteration undertaken by PEST; an *N* value of zero corresponds to initial parameter values as provided in the PEST control file on which basis traversal of the Pareto front was undertaken.

Objective function values corresponding to any parameter set are available in ASCII form in the Pareto objective function file (i.e. file *case.pod*) written by PEST. The parameter set corresponding to iteration *N* recorded in binary form in the Pareto parameter data file (this being the *N*+1'th parameter set recorded in that file as counting begins at zero) corresponds to the *N*+1'th set of objective functions recorded in the Pareto objective function file. (The first objective function in that file corresponds to initial parameter values too.) Thus a point of interest on the Pareto front can be selected using a graph based on data residing in the Pareto objective function file; the parameter set corresponding to that point can then be extracted from the Pareto parameter data file using PPD2PAR.

The parameter set obtained by running PPD2PAR for a particular *N* corresponds to that residing in the parameter value file *case.par.N* written by PEST as it runs in Pareto mode.

14. Observation Re-referencing

14.1 Introduction

The use of so-called "observation re-referencing" adds considerable flexibility to the way that PEST can run a model. It allows PEST to run different variants of a model for different purposes. As such, it is an expansion of PEST's multiple command line functionality documented in section 12.3 of this manual. However observation re-referencing goes well beyond this, allowing at least the following strategies to be employed in model calibration and uncertainty analysis.

- Use of a simple model, or multiple simple models, in place of a complex model for the purpose of finite-difference derivatives calculation. Meanwhile parameter upgrades are tested and applied on the complex model.
- The ability to alter initial model conditions (for example initial heads in a steady state groundwater model) as parameters are upgraded. Where initial conditions can be made to approximate final conditions, the convergence time of iterative model solvers can be increased dramatically if these conditions are re-calculated by the model on the basis of most recently upgraded parameters.

14.2 General Principles

On most occasions of its deployment, PEST calculates derivatives of model outputs with respect to adjustable parameters using finite differences. In some cases it may be possible for a different model to be employed for the purpose of derivatives calculation from that which is used for the purpose of testing and upgrading parameters. This may be desirable where a simplified "surrogate model" is run for the purpose of derivatives calculation, and this model has a short run time compared with that of the model which is actually being calibrated. If this strategy is attempted, however, certain problems may arise. These will now be discussed.

The difference equation through which the derivative of observation *j* with respect to parameter *i* is calculated can be written as

$$\frac{\partial o_j}{\partial p_i} \approx \frac{o_j^{vi} - o_j^{r}}{p_i^{v} - p_i^{r}} \qquad (14.2.1)$$

In equation 14.2.1 p^{r}_i is the base value (i.e. the current reference value) of parameter p_i while o^{r}_j is the value of the model output corresponding to observation o_j calculated by the model using the base parameter set. p^{v}_i is the value of parameter *i* when incrementally varied for the purpose of derivatives calculation while o_j^{vi} is the model-calculated value of observation *j* when parameter *i* is thus varied.

Normally o^{r}_j is calculated by PEST using either the initial model run, or during the parameter upgrade process where different model runs are undertaken on the basis of different trial parameter sets calculated using different Marquardt lambdas. In the latter case the best upgrade is selected and new parameters are based on that; meanwhile model outputs calculated on the basis of the best parameter set become the base model outputs or "reference model outputs" used in the above equation for finite-difference derivatives calculation for the next PEST iteration.

A problem arises if a different model (for example a simpler and faster-running model) is to

be employed for derivatives calculation from that which is used for testing parameter upgrades. In this case reference model outputs calculated using the complex model are not appropriate for use in the finite-difference derivatives equation when applied to outputs of the simple model. The above equation is only valid when the same model is used to calculate both o^{vi}_j and o_j^r. Hence, for finite-difference-calculated derivatives to have integrity in this case, the simple model must be run especially to calculate o^r_j using the reference parameter set. This requires a special model run to be undertaken at the beginning of each iteration. This special model run is indeed carried out if "observation re-referencing" is activated.

In more complex cases where more than one version of the model is used for the purpose of derivatives calculation (for example if multiple simple models are employed, each used for the calculation of derivatives with respect to a different subset of parameters), then an observation re-referencing model run must be undertaken for each version of the model that is so employed.

Using the multiple command line functionality described in section 12.3 it is possible to employ different model versions for calculation of finite-difference derivatives with respect to different parameters. However use of this functionality on its own requires that the different models employed for this purpose are all components of the one large model, and that the same set of reference model outputs (namely those calculated by the total model) are applicable to all of them. Under these circumstances observation re-referencing is not required. However with implementation of observation re-referencing in conjunction with multiple model commands, it is no longer required that models used in finite-difference derivatives calculation produce the same outputs as the main model if supplied with the same set of reference parameter values.

14.3 Two Observation Re-Referencing Modes

14.3.1 Mode 1

The first mode of PEST's observation re-referencing functionality is automatically implemented if observation re-referencing functionality is activated and only one command is used to run the model. Thus the NUMCOM variable must be set to 1 in the "control data" section of the PEST control file (or can be omitted altogether). At the same time, if the DERCOM variable appears in the "parameter data" section of the PEST control file, it must be assigned a value of 1 for each adjustable parameter.

If NUMCOM is set to 1 (or omitted) a single model command is provided in the "model command line" section of the PEST control file. As usual, this command may refer to an executable program, or to a batch (or script) file which cites many different executable programs which are run in succession. If observation re-referencing is activated under these circumstances, then you must prepare not just the program or batch file cited in the "model command line" section of the PEST control file; you must supply two other programs or batch files as well. These are not mentioned by name in the PEST control file. However their names are assumed to be formed from the command cited in the PEST control file by addition of an "r_" prefix and a "d_" prefix to that name. Thus if the command cited in the "model command line" section of the PEST control file is *command.bat*, then PEST will, at various stages of its execution, issue three different commands to the operating system in order to run the model, these being *command.bat*, *r_command.bat* and *d_command.bat*.

These different model commands are used by PEST in the following manner.

- The command provided to PEST in the "model command line" section of the PEST control file (*command.bat* in the above example) is used by PEST to run the model for the first time, and to undertake attempted upgrades of parameters on the basis of different Marquardt lambdas. Outputs of this model are used to calculate the objective function that is written to the screen and recorded on PEST output files. These are also the model outputs that are recorded in the run record file and in the residuals file.
- The command with the "d_" prefix (*d_command.bat* in the above example) is used for the purpose of finite-difference derivatives calculation. This command is issued at least once for each adjustable parameter during the Jacobian calculation phase of all iterations.
- The command with the "r_" prefix (*r_command.bat* in the above example) is used by PEST as it undertakes one additional run of the model during each iteration. This run is undertaken just prior to running the model at least once for each adjustable parameter for the purpose of filling the Jacobian matrix (i.e. for derivatives calculation). The parameters that are employed for this single run are the current reference values of the parameters - these having just been upgraded during the previous iteration. Model outputs calculated on the basis of this command are used as model output reference values for the purpose of derivatives calculation. Thus this command is used to compute o^r_j featured in equation 14.2.1 for all j.

In many calibration circumstances the batch file which uses the "d_" prefix will have identical contents to that which uses the "r_" prefix. Thus reference model outputs are computed using the same version of the model as are outputs calculated using incrementally varied parameters for the purpose of derivatives calculation, this ensuring the integrity of finite-difference derivatives calculation. On other occasions the re-referencing command may be more complex than this. For example if it is undertaken for the purpose of providing a revised set of initial heads for the groundwater model discussed above (and the model employed for derivatives calculation is the same as that used to compute the objective function), the "r_" batch file may invoke commands which implement the following processing sequence:

- Computation of model outputs on the basis of current reference parameters values (which presumably have just been updated).
- Copying of domain-wide heads calculated on the basis of newly-upgraded reference parameters to an initial heads file.
- (If runs are parallelised), distribution of this initial heads file to all slave directories.
- Re-running the model once (still using newly-upgraded reference parameters) so that reference model outputs o^r_j are calculated under exactly the same numerical circumstances as those that are just about to be employed for calculation of model outputs on the basis of perturbed parameter values (this involving use of the new initial heads file).

14.3.2 Mode 2

This mode of observation re-referencing is automatically activated when PEST employs multiple model command lines. Thus NUMCOM is set to a number greater than 1 in the "control data" section of the PEST control file, and DERCOM variables associated with different parameters refer to different command lines. At the same time, more than one command is cited in the "model command line" section of the PEST control file.

In this case, as discussed in section 12.3, when PEST undertakes the initial model run, and

when it undertakes parameter upgrades, it employs the first model command cited in the "model command line" section of the PEST control file, this being the command with index 1. When running the model for the purpose of derivatives calculation it can use any of the commands cited in the "model command line" section, this depending on the DERCOM value associated with each parameter.

When observation re-referencing is activated for a PEST case involving multiple command lines this protocol is still followed. However PEST undertakes a number of additional model runs at the beginning of each iteration, just prior to undertaking model runs for the purpose of derivatives calculation. These are observation re-referencing runs.

Suppose that the commands listed in the "model command line" section of the PEST control file are *command1.bat*, *command2.bat*, *command3.bat*, etc. If observation re-referencing is activated, PEST will employ the commands *r_command2.bat*, *r_command3.bat* etc. to undertake observation re-referencing runs just before undertaking finite-difference Jacobian calculation using the commands *command1.bat*, *command2.bat*, *command3.bat*, etc. as appropriate for each parameter. It is the user's responsibility to ensure that the appropriate "r_" prefixed commands can be executed; normally this will entail construction of appropriate batch files.

Note the following.

- An observation re-referencing model run is NOT undertaken for the first command employed in the "model command line" section of the PEST control file (i.e. *command1.bat* in the above example) because this same command was employed for the first model run and for the purpose of testing parameter upgrades. Reference model outputs calculated using this command are thus available, and are appropriate for finite-difference derivatives calculation for those parameters (if any) for which the first model command is run on the basis of incrementally varied parameters (i.e. for which the parameter-specific DERCOM setting is 1).
- Where a particular model command is employed to calculate finite-difference derivatives for a particular parameter, the reference model outputs employed in the finite-difference equation are those computed using the corresponding "r_" prefixed model command.
- If no parameters are assigned to a particular model command (i.e. if no parameter DERCOM value provides the index for that command), then that command is not employed in derivatives calculation. An observation re-referencing run is not then undertaken using the pertinent "r_" prefixed command.

14.4 Activating Observation Re-Referencing

Observation re-referencing is activated by adding the string "obsreref" to the fourth line of the "control data" section of the PEST control file. This is the same line as that on which the NTPLFLE, NINSFLE, PRECIS, DPOINT variables, and the optional NUMCOM, JACFILE and MESSFILE variables appear. The "obsreref" string can be placed anywhere on the line. Observation re-referencing can be de-activated (the default condition) by omitting this string, or by citing the string "noobsreref" on this line.

14.5 SVD-Assist and Observation Re-referencing

SVDAPREP can accommodate PEST control files in which observation re-referencing is

activated. However it can only accommodate mode 1 of observation re-referencing. (Implementation of mode 2 would require that different commands be used to run the model for different parameters when undertaking SVD-assisted parameter estimation. However this is not possible as super parameters employed by PEST when undertaking SVD-assisted parameter estimation are each composed of many different base parameters. The relationships between parameters and model runs that are defined in a base parameter PEST control file cannot apply to super parameters.)

When SVDAPREP is run on the basis of a base PEST control file in which observation re-referencing is activated, it looks for "r_" and "d_" prefixed batch files that match the batch file supplied in the "model command line" section of the PEST control file. Then, in addition to the *svdabatch.bat* batch or script file, SVDAPREP writes files *r_svdabatch.bat* and *d_svdabatch.bat* by adding the commands to run PARCALC and, if necessary PICALC, to the contents of these existing "r_" and "d_" prefixed batch files prior to any other commands cited in these files. The new PEST control file which is written by SVDAPREP instructs PEST to activate observation re-referencing. This file is immediately ready for running by PEST.

14.6 Parallelisation

BEOPEST can be employed to run PEST on the basis of a PEST control file that implements observation re-referencing of either mode (including an SVD-assist PEST control file wherein mode 1 of observation re-referencing is activated). However Parallel PEST cannot be used under these circumstances.

A special item of observation re-referencing functionality is available for use with BEOPEST. This is activated by adding a suffix to the "obsreref" keyword in the "control data" section of the PEST control file. Suppose this is written as "obsreref_10". Then the BEOPEST run manager will do two things. Firstly, regardless of the number of slaves at its disposal, it will separate each Jacobian parcel of model runs into two sub-parcels. For the first parcel it will initiate only as many model runs as are required for observation re-referencing and will initiate no other model runs until these runs are complete; thus one model run will be initiated if mode 1 observation re-referencing functionality is operative, while normally more than this will be initiated if mode 2 observation re-referencing functionality is operative. BEOPEST will not activate any model runs for the purpose of derivatives calculation until all of these re-referencing runs have been completed. Then it will pause for *N* seconds, where *N* is the integer followed the "obsreref" keyword and ensuing underscore; this is 10 in the above example. As soon as *N* seconds have elapsed, Jacobian model runs are initiated, with maximum use made of all available slaves. This allows a user to implement the strategy discussed above whereby re-referencing model runs allow initial simulator conditions to be set for Jacobian model runs. If the model batch file(s) for the former model run(s) include commands for copying of initial condition files to the working directories of all slaves, this strategy ensures that none of the Jacobian model runs which use these initial conditions are started before the new initial conditions become available. It achieves this by giving the network time to complete the copying operation before the Jacobian runs are actually commenced.

14.7 Restarting

Normal re-starting functionality is operative when observation re-referencing is employed.

Use of the "/i" and "/p1" command line switches is also permitted. However where BEOPEST is run and a partitioning of model runs is activated in the manner discussed above, then the "/p1" switch is not allowed. The mixture of run types in the first bundle of parallel runs would be simply too hard to manage.

15. Large Numbers of Parameters

15.1 Introduction

This chapter discusses a number of devices that are offered by PEST for calibration of models that employ tens of thousands of parameters. PEST can performed well in these settings, but will not do so unless at least some of the measures outlined herein are adopted.

15.2 Derivatives Calculation

Where a large number of parameters is being estimated (in the tens of thousands) it becomes mandatory that derivatives be calculated by the model itself using algorithms such as adjoint techniques which have been developed specifically for this purpose. Use of the finite-difference methodology for calculation of derivatives becomes impossible under these circumstances for two reasons. These are

- the large number of model runs that this requires; and
- the lack of numerical precision that accompanies finite-difference calculation of derivatives with respect to parameters whose individual sensitivities may be low because of their large collective number.

As was discussed in chapter 12 of this manual, a model can supply self-calculated derivatives to PEST using an external derivatives file. Disk space and reading time can be saved if the compressed storage protocol is used for this file, and if binary, rather than ASCII storage is employed.

If using the compressed option for storage of external derivatives, and if using compressed Jacobian matrix storage within PEST itself (see below), maximum efficiency in reading the external derivatives file is achieved if indexed derivative matrix elements listed in the external derivatives file are cited in the same order as that in which they are stored internally by PEST in its Jacobian matrix. Storage order within the PEST Jacobian matrix is column by column; hence indexing in the external derivatives file should progresses firstly by observation, and then by parameter.

15.3 Versions of PEST

Where problem sizes are large, use 64 bit versions of PEST, Parallel PEST and BEOPEST (and of PEST support utilities for which 64 bit versions are provided). Execution is generally faster. Much more memory can be addressed.

15.4 Compressed Internal Jacobian Storage

15.4.1 Concepts

Where large numbers of parameters are estimated, the memory required for storage of the Jacobian matrix may become impossibly high. The dimensions of the Jacobian matrix are $(n_o+n_{p)}\times m$ where n_o is the number of observations, n_p is the number of prior information equations and m is the number of adjustable parameters. In highly parameterised inversion contexts, many sensitivities of model outputs with respect to parameters can be close to zero. Furthermore, the use of Tikhonov regularisation is often accompanied by large numbers of

zero-valued Jacobian matrix elements. Each regularisation constraint normally occupies one row of the Jacobian matrix. There is normally at least one such constraint for every estimated parameter – but sometimes many more than this. Often most of the elements of each such row of the Jacobian matrix are zeroes, as each regularisation constraint may involve either one parameter in an equality relationship, or two parameters in a differencing relationship. If follows that a data storage mechanism which dispenses with the need to explicitly store element values of zero will realize large reductions in PEST's memory requirements.

15.4.2 The MAXCOMPDIM Control Variable

To address this issue, PEST supports compressed internal storage of the Jacobian matrix. This compressed storage mechanism is supplemented by programming within PEST that accesses elements of the compressed Jacobian matrix in ways that are most efficient for the types of Jacobian matrix calculations normally undertaken by PEST. There is some loss in inversion speed incurred by the use of compressed Jacobian matrix storage; however the cost of compression has been reduced as much as possible.

Jacobian matrix compression is activated through use of an optional variable which resides on the third line of the "control data" section of the PEST control file. This variable is named MAXCOMPDIM; see figure 15.1 for its location. If MAXCOMPDIM is omitted from the PEST control file, or is set to 1 or less, no Jacobian compression takes place. If it is set to greater than 1, the vector which holds the compressed form of the Jacobian matrix is dimensioned as MAXCOMPDIM, and Jacobian compression takes place. If, in the course of its execution, PEST discovers that MAXCOMPDIM has not been set large enough to hold the compressed Jacobian matrix, it will cease execution with an appropriate error message.

```
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM] [DERZEROLIM]
NTPLFLE NINSFLE PRECIS DPOINT [NUMCOM JACFILE MESSFILE] [OBSREREF]
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE] [LAMFORGIVE] [DERFORGIVE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND] [ABSPARMAX]
PHIREDSWH [NOPTSWITCH] [SPLITSWH] [DOAUI] [DOSENREUSE] [BOUNDSCALE]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN] [PHIABANDON]
ICOV ICOR IEIG [IRES] [JCOSAVE] [VERBOSEREC] [JCOSAVEITN] [REISAVEITN] [PARSAVEITN] [PARSAVERUN]
```

Figure 15.1 "Control data" section of the PEST control file with the optional MAXCOMPDIM variable highlighted.

The exact manner in which PEST stores the compressed Jacobian matrix depends on the value that is supplied for MAXCOMPDIM. If it is possible, PEST tries to subdivide the Jacobian matrix into two submatrices. The first (with dimensions $n_o \times m$) pertains only to observations; the second (with dimensions $n_p \times m$) pertains only to prior information. Not only is the second submatrix normally much sparser than the first. Its elements need to be calculated only once.

If the user-supplied value for MAXCOMPDIM is greater than $n_o \times m + n_{nz} + 10$ where n_{nz} is the number of non-zero elements in the prior information submatrix of the Jacobian matrix, then PEST will store the observation submatrix of the Jacobian matrix in standard form (to allow easy access to elements of this matrix). Meanwhile it stores the prior information submatrix of the Jacobian matrix in compressed format wherein internal indexing is such as to allow rapid access to neighbouring nonzero elements where nonzero elements are very sparse. In many inversion contexts n_{nz} is easily calculated. For example if there are n_p prior information equations, and each such equation cites n_c parameters, then n_{nz} is readily calculated as $n_p \times n_c$.

PEST knows soon after commencement of execution whether it can adopt this second

protocol for storage of the Jacobian matrix. If it cannot, then it adopts the first protocol wherein compressed storage is implemented for the entirety of the Jacobian matrix. However this brings with it the problem that PEST does not have foreknowledge of how many zero-valued elements the observation component of the Jacobian matrix will contain. This can only be known as the Jacobian matrix is actually filled, either through the undertaking of model runs for the purposes of finite-difference derivatives calculation or through reading a model-produced external derivatives file. Under the former circumstances, a large number of model runs need to be undertaken before PEST ceases execution with an error message that MAXCOMPDIM needs to be set higher.

If PEST does indeed inform you that MAXCOMPDIM needs to be set higher, then the PEST control file must be edited accordingly and PEST re-started. Fortunately execution of PEST can then be re-commenced with the "/s" or "/d" switches (depending on whether it is being run in parallel or serial mode). Previous model runs are therefore not wasted.

15.4.3 The DERZEROLIM Control Variable

A PEST variable named DERZEROLIM can optionally follow MAXCOMPDIM on the third line of the "control data" section of the PEST control file. This should be entered as a low number or zero. A finite-difference derivative is assumed to be zero (and hence not stored in the compressed Jacobian matrix) if its absolute value is less than this number. This allows numerical noise in finite-difference derivatives incurred, for example, by problematic simulator solver convergence, to be filtered out to at least some extent at the same time as it reduces Jacobian matrix storage requirements. Note that the DERZEROLIM threshold is not applied to externally-calculated derivatives; nor is it applied to finite-difference derivatives unless MAXCOMPDIM is greater than one, and hence compressed Jacobian storage is activated.

15.5 Ordering of Observation Groups

If regularisation constraints are supplied through observations rather than prior information, then potentially large savings in the time required to perform certain matrix operations required by the regularisation process can be achieved if all regularisation information follows all other observations in the "observation data" section of the PEST control file. That is, all observations belonging to observation groups whose names begin with "regul" should be supplied together; furthermore these should follow observations pertaining to all other observation groups. PEST detects this situation when it reads the PEST control file; subsequent manipulation of observation data takes numerical advantage of this.

15.6 Linear Regularisation Constraints

Regularisation constraints can be supplied through observations, through prior information, or through both of these mechanisms. Prior information relationships are always linear. Regularisation constraints supplied as observations (for which the current value of pertinent relationships is calculated by the model), can be linear or nonlinear; in either case, derivatives of these relationships with respect to adjustable parameters are re-evaluated by PEST (or supplied by the model) during each iteration of the inversion process.

If regularisation information is entirely linear, there are many matrix operations carried out as part of PEST's regularisation functionality which do not need to be repeated from iteration to iteration. If repetition of these calculations can be avoided in parameter estimation contexts

involving many regularisation constraints, significant gains in computational efficiency can be made. The user can inform PEST that all regularisation constraints are linear through an optional control variable supplied in the "regularisation" section of the PEST control file. The name of this variable is LINREG; it should be supplied as either "linreg" or "nonlinreg". Figure 15.2 shows the location of this variable within this section of the PEST control file, while figure 15.3 illustrates its use. LINREG can be placed anywhere on the third line of the "regularisation" section of the PEST control file after the value of the WFMAX variable. As discussed in section 9.2 of this manual, its placement can be interchanged with that of the optional REGCONTINUE variable.

```
* regularisation
PHIMLIM  PHIMACCEPT FRACPHIM
WFINIT  WFMIN  WFMAX [LINREG]
WFFAC  WFTOL IREGADJ
```

Figure 15.2 "Regularisation" section of a PEST control file showing the location of the optional LINREG variable.

```
* regularisation
  0.00001   0.00002     0.1
  1.E-2  1.0e-10  1.0e10 linreg
  1.5    1.0e-2  1
```

Figure 15.3 Example of the use of the LINREG variable.

If a value for LINREG is not supplied in a PEST control file, PEST uses the default value of "nonlinreg". However if all regularisation constraints are supplied as prior information PEST sets LINREG to "linreg" automatically.

It is very important to take account of the transformation status of parameters when assigning a value to LINREG. Whether a model supplies derivatives to PEST itself using PEST's external derivatives functionality, or whether PEST calculates derivatives by finite differences, a value is calculated for the derivative of each model output with respect to each parameter. For a particular parameter, this derivative is modified before being assigned to the respective element of the Jacobian matrix if the parameter is log-transformed. It is the Jacobian matrix that must be considered when classifying a relationship as linear or nonlinear. Thus, if a model calculates derivatives with respect to a parameter, and that parameter is log-transformed, the linear/nonlinear status of model-calculated derivatives does not determine the linearity status of the final derivative (this being the respective element of the Jacobian matrix). It is the derivative with respect to the log of the parameter that matters in assigning a value to LINREG in this case.

15.7 Trading Memory for Functionality

The optional MEMSAVE variable, situated within the "regularisation" section of the PEST control file, can be used to turn on, and turn off, certain types of memory conservation. MEMSAVE is a character variable which must be supplied as either "memsave" or "nomemsave". Figure 15.4 shows the location of this variable in the PEST control file; it follows the optional FRACPHIM variable on the second line of the "regularisation" section. If FRACPHIM is omitted, MEMSAVE must follow PHIMACCEPT.

```
* regularisation
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE]
WFINIT  WFMIN  WFMAX  [LINREG] [REGCONTINUE]
WFFAC  WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]
```

Figure 15.4 Location of the optional MEMSAVE variable.

If MEMSAVE is supplied as "memsave", then PEST sets the ICOV, ICOR and IEIG variables (in the "control data" section of the PEST control file) to zero. Hence PEST does not compute a covariance matrix, nor leave a trail of *case.mtt* files. However this functionality is probably already automatically disabled in very highly parameterized cases; PEST disables it automatically if singular value decomposition or LSQR are used as solution mechanisms for the inverse problem, and/or if Tikhonov regularization is supplied.

The LINREG variable is internally set to "nonlinreg" if MEMSAVE is supplied as "memsave". Furthermore, this "nonlinreg" setting applies even if regularisation is supplied purely as prior information. While this can lead to significant savings in memory, it can also lead to significant run-time penalties where parameter numbers are very large. Hence it should be avoided unless absolutely necessary.

15.8 Accelerated Input of Prior Information

When estimating values for a large number of parameters (for example ten thousand or more parameters), the handling of prior information that may provide Tikhonov constraints for these parameters may become problematical if compressed Jacobian storage is employed (i.e. if the MAXCOMPDIM variable in the "control data" section of the PEST control file is set to greater than 1). As has already been discussed, the Jacobian submatrix which holds this prior information is likely to be sparse; hence considerable benefits are to be gained from storing it in compressed form. However storage and retrieval of items that are stored in compressed format requires calculations to be carried out which may increase the computational burden of the overall inversion process.

Considerable computational advantage can be gained if prior information is provided to PEST in the same order as that in which it is stored internally in the compressed Jacobian array. Elements of the compressed Jacobian matrix are stored in row order, with zero-valued elements excluded. That is, elements are stored in the order $\mathbf{J}(i,j)$, $\mathbf{J}(i+1,j)$, $\mathbf{J}(i+2,j)$ etc., with the jump made to column $j+1$ when all non-zero values in column j have been stored. The row number is the observation (including prior information) number (i in this example), while the parameter number is the column number (j in this example).

In the "prior information" section of the PEST control file, the user supplies prior information in the form of equations. If more than one parameter is cited in any of these equations, the above ordering is not respected. Where many prior information equations are supplied in order to provide regularisation constraints for many estimated parameters, PEST may require a considerable amount of time to read and store this prior information when compressed Jacobian matrix storage is employed.

In order to overcome this problem, PEST provides an alternative means through which prior information can be supplied. This is referred to as "indexed prior information" herein.

PEST is informed that prior information is provided in indexed form in the "prior information" section of the PEST control file if the NPRIOR variable in the "control data" section of the PEST control file is supplied as negative. As usual, the absolute value of NPRIOR must indicate the number of prior information equations that are featured in the

current inverse problem.

When the indexed prior information protocol is adopted, the "prior information" section of the PEST control file is subdivided into two sections. There are no headers between these subsections. The first subsection should contain NPRIOR lines of data. Each line of this subsection must contain four entries. These are as follows.

- The name of the prior information equation (12 characters or less without quotes or blanks); this is the PILBL variable.
- The "observed value" of the prior information equation (a real number); this is the PIVAL variable.
- The weight associated with the prior information equation (a real number); this is the WEIGHT variable.
- The observation group to which the prior information equation belongs (12 characters or less citing a group that has already been named in the "observation groups" section of the PEST control file); this is the OBGNME variable.

Part of this subsection of the "prior information" section is exemplified in figure 15.5.

```
* prior information
pr_r1  0.0  1.0 regul_row
pr_r2  0.0  1.0 regul_row
pr_r3  0.0  1.0 regul_row
pr_r4  0.0  1.0 regul_row
pr_r5  0.0  1.0 regul_row
pr_r6  0.0  1.0 regul_row
pr_r7  0.0  1.0 regul_row
pr_r8  0.0  1.0 regul_row
pr_r9  0.0  1.0 regul_row
pr_r10 0.0  1.0 regul_row
pr_r11 0.0  1.0 regul_row
pr_r12 0.0  1.0 regul_row
pr_r13 0.0  1.0 regul_row
pr_r14 0.0  1.0 regul_row
pr_r15 0.0  1.0 regul_row
etc
```

Figure 15.5 Part of the first subsection of the "prior information" section of a PEST control file in which prior information is supplied in indexed format.

Following NPRIOR items supplied as above is the second subsection of the "prior information" section of the PEST control file. This subsection begins with a line containing a single integer, this specifying the number of lines to follow. This integer must equal or exceed NPRIOR; it will be referred to as NUMINDEX herein.

Each of the following NUMINDEX lines must contain three entries. These are, in order:

- a column number of the Jacobian matrix (an integer);
- a row number of the Jacobian matrix (an integer);
- the value of the element of the Jacobian matrix corresponding to the nominated row and column numbers.

A Jacobian matrix column number is obtained by counting parameters in order of their appearance in the "parameter data" section of the PEST control file; fixed and tied parameters are omitted from this count. A row number is obtained by counting first observations, and then prior information equations, in order of their appearance in the PEST control file. For prior information equations, the row number must exceed NOBS (where NOBS is the number of observations featured in the PEST control file), and must be less than or equal to

NOBS+NPRIOR. It is the user's task to calculate these indices him/herself; normally a PEST control file which features indexed prior information will be written by utility software which undertakes these calculations as part of its processing services.

The value of the Jacobian element is the factor by which the parameter (or the log of the parameter if it is log-transformed) is multiplied in the respective prior information equation. This is equivalent to the sensitivity of the outcome of the equation to the parameter (or log of the parameter).

An example of the start of the second subsection of an indexed "prior information" section of a PEST control file is provided in figure 15.6.

```
89544
    1      1090    1.000000
    1     16040    1.000000
    1     30990    1.000000
    2      1090   -1.000000
    2      1091    1.000000
    2     16065    1.000000
    2     31012    1.000000
    3      1091   -1.000000
    3      1092    1.000000
    3     16090    1.000000
    3     31034    1.000000
    4      1092   -1.000000
    4      1093    1.000000
    4     16115    1.000000
    4     31056    1.000000
    5      1093   -1.000000
    5      1094    1.000000
    5     16140    1.000000
```

Figure 15.6 Start of the second subsection of the "prior information" section of a PEST control file in which prior information is supplied in indexed format.

The fact that prior information is supplied in indexed format does not, of itself, guarantee fast compressed storage of this information, though it does help a little. What makes all the difference is the order in which items comprising the second subsection of the indexed "prior information" section of the PEST control file are supplied. These should be supplied in the same order as that in which they will be stored internally by PEST. Thus the observation number must vary most quickly (second column of the above figure) while the parameter number (first column of the above figure) must vary most slowly; numbers in the first column should never decrease.

15.9 Accelerating Input of External Derivatives

As described in section 15.8, if PEST stores the Jacobian matrix internally in compressed form then it is stored in column order; that is, all elements pertaining to the first parameter are stored first, then for the second, etc.; zero-valued elements are omitted. As was explained in section 15.2, if derivatives are supplied in an external derivatives file, and if the compressed protocol is employed for this file, then the time required to read and process the elements of the derivatives matrix may be reduced if these elements are supplied in the same order as that in which PEST stores elements of the compressed Jacobian matrix. This will especially be the case on the first occasion that the matrix is read, for on that occasion a constant rearrangement of compressed storage elements within PEST's memory will be the inevitable

result of supplying these elements in random order. However no such re-arrangement is necessary if all derivative matrix elements for parameter number 1 are supplied first (in order of increasing observation number), followed by all elements for parameter 2, etc.

Recall from section 15.4.2 however, that problems with Jacobian matrix access can also be reduced if the value for MAXCOMPDIM is chosen wisely. Recall that a suitable choice for MAXCOMPDIM may promulgate storage of the observation submatrix of the overall Jacobian submatrix in the more easily accessible standard form. However memory resources must be large enough to allow this. If they are not, then loss of efficiency incurred through compressed Jacobian element rearrangement on the first occasion on which the external derivatives file is read by PEST will occur if prior information is used in the inversion process (for derivatives pertaining to prior information equations are stored at the lower rows of the Jacobian matrix). This problem can be overcome through the use of the indexed prior information protocol discussed in the previous subsection. However if you do not wish to do this, you may wish to consider supplying all regularisation constraints (even linear constraints) as "observations" (computed by the model) rather than as prior information equations in the "prior information" section of the PEST control file, for then the respective elements of the Jacobian matrix can be supplied in correct order. When doing this, care must be taken when supplying derivatives for log-transformed parameters. For example the derivative with respect to *par1* of the relationship

```
log(par1) - log(par2) = 0
```

is 1.0/*par1*/2.303. If the parameter *par1* is log-transformed by PEST, this relationship becomes linear internally to PEST.

15.10 Other Savings

Where parameter numbers are large, the reading and writing of information pertaining to the inversion process can be a lengthy matter which can add to the time required for solution of the inverse problem. In some instances the following measures can accrue considerable savings in PEST speed (and in PEST's disk footprint).

- Set the RSTFLE variable to "norestart";
- Set JCOSAVE to "nojcosave";
- Set VERBOSEREC to "noverboserec";
- Set ICOV, ICOR, IEIG and IRES to 0.

16. PEST-Compatible Global Optimisers

16.1 CMAES_P

16.1.1 General

CMAES_P is a "PEST compatible" implementation of the powerful and robust CMA-ES global optimisation scheme. For details of this scheme see, for example, Hansen and Ostermeier (2001) and Hansen et al (2003). See Bayer and Finkel (2007) for an application of the algorithm to groundwater contaminant capture. See also Nikolaus Hansen's web pages at

https://www.lri.fr/~hansen/

for further details, including a tutorial as well as MATLAB and OCTAVE source code. Although a brief outline of the algorithm will be presented below, it is strongly suggested that you learn something about this algorithm from these sources, so that you are equipped with the knowledge necessary to adjust various control variables in order to meet the demands of different problems.

Unlike PEST, CMAES_P does not require derivatives of model outputs with respect to adjustable parameters for implementation of its optimisation algorithm. Thus it can be employed where model outputs show "numerical granularity" due to model numerical solution instability, or where the model is highly nonlinear and/or the objective function surface shows local minima at various scales. Where model derivatives have integrity, PEST's performance will almost certainly be superior to that of CMAES_P (in terms of the number of model runs required to minimise an objective function). Where model derivatives do not have integrity, however, CMAES_P's performance may be superior to that of PEST.

CMAES_P was built from version 2.5 of CMA-ES. I would like to point out, however, that any problems, imperfections and inefficiencies encountered in using CMAES_P were probably introduced by myself rather than being characteristics of the algorithm. I apologize if I have inadvertently done this.

16.1.2 The Algorithm in Brief

"CMA" stands for "covariance matrix adaptation". "ES" stands for evolutionary strategy. The algorithm employed by CMAES_P is sometimes given the longer name "CMA-ES with rank μ update and weighted recombination" or simply "(μ, λ) CMA-ES".

The CMA-ES algorithm employs an iterative procedure to minimise an objective function which is dependent on m adjustable parameters. In each iteration of this procedure λ random realisations of m-dimensional parameter vectors are generated, and the objective function is computed for each (this requiring one model run in each case). Normally λ (also referred to as the "population size") is considerably smaller than m. The m-dimensional covariance matrix used to generate these parameter realisations evolves through the optimisation process in response to what is learned about the objective function surface by sampling it in this way. This maintains efficiency of the process, for as the minimum of the objective function is approached, the search radius about that minimum for possible lower objective functions is reduced in order to exclude from the m dimensional search area portions of parameter space in which it is extremely unlikely that the global minimum lies. On the other hand, the fact that the search process is random reduces the chances of being trapped in a local minimum of

the objective function.

Once the λ model runs have been carried out, the parameter sets giving rise to the μ lowest objective functions are selected, and a new "mean" parameter set is calculated through formulating a weighted average of the corresponding μ parameter value sets (μ is sometimes referred to as the "number of parents"). Weights can be assigned as "super-linear", "linear", or "equal". In the first two cases, greater weight is given to parameters that give rise to lower objective functions, this often leading to faster reduction of that function. As will be discussed below, the CMAES_P default is "super linear" (in which lower objective functions are weighted more heavily than in the "linear" option); though in many cases "linear" works at least as well.

Once a new average set of parameter values has been computed in this fashion, the next iteration begins. Because random parameter realisations are generated to be symmetrical about this mean, there is a tendency of the objective function to fall as iterations proceed.

An extremely important component of the CMA-ES methodology, however, is its capacity to adapt the parameter covariance matrix that forms the basis for generation of random parameter samples as the optimisation process progresses. Adaptation takes place on the basis of wisdom gained from both the current iteration, and from previous iterations, the mix between the two being set internally by the CMA-ES algorithm. See the references cited above for further details.

The CMA-ES algorithm goes to considerable lengths in its handling of parameter bounds. Bounds are never violated, for if any member of a random parameter set violates its individual bound, it is adjusted to respect it. In addition to this the tendency to generate parameters which violate bounds is mitigated through adding a penalty to the objective function (internally to the code) as this occurs; this in turn affects the evolving parameter covariance matrix on the basis of which further parameter realisations are generated.

16.1.3 The CMAES_P Implementation of CMA-ES

Introduction

CMAES_P reads a PEST input dataset, including the PEST control file and template and instruction files cited therein. Like PEST, it communicates with a model through that model's own input and output files. Like PEST, it runs a model many times, maximising the fit between model outputs and field measurements recorded in the PEST control file (unless it is asked to minimise a single model output which is assumed to represent a model-calculated objective function – see below).

Features in Common with PEST

CMAES_P has the following features in common with PEST.

1. The PEST input dataset read by CMAES_P can include prior information, multiple observation groups, and can use observation covariance matrices instead of observation weights. The objective function calculated by CMAES_P is exactly the same as that calculated by PEST.
2. Model runs can be undertaken in serial, or in parallel. The parallel run protocol is the same as that provided by Parallel PEST. (There is presently no "BEOCMAES_P".) If parallelisation is implemented, CMAES_P must be provided with a run management file. This has the same filename base as the PEST control file, but possesses an

extension of "*.rmf*". Hence if the PEST control file is named *case.pst*, the run management file must be named *case.rmf*. The history of communication between CMAES_P and its slaves is recorded in a run management record file. This is named *case.rmr*.

3. Like PEST, CMAES_P records the progress of the optimisation process both to the screen and to a run record file. The latter is named *case.rec*.
4. At any stage of the optimisation process, and at the end of this process, best parameters computed to date can be found in a "parameter value file" named *case.par*.
5. A CMAES_P run can be terminated by issuing the PSTOP or PSTOPST command in another window. Its execution can be paused and unpaused using the PPAUSE and PUNPAUSE commands.

Parameter Estimation and Optimisation

PEST minimises an objective function computed as the sum of squared weighted residuals between model outputs and field or laboratory measurements. So too does CMAES_P. However use of the CMA-ES algorithm is not restricted to an objective function computed in this manner, for it can be employed to minimise an objective function calculated in any way whatsoever (even an objective function whose minimum value is less than zero). While code internal to CMAES_P computes an objective function in the same way that PEST does, CMAES_P provides the option of minimising a single model-generated number (this presumably being a model-calculated objective function). This option becomes available when the PEST control file read by CMAES_P contains a single observation and no prior information. In this case CMAES_P asks the user whether the single model output corresponding to the observation is to be treated as an objective function to be minimised, or whether it should be matched to the corresponding measurement in the PEST control file, thereby formulating a one-term least squares objective function to be minimised. In the former case, CMAES_P does not calculate an objective function internally, turning its attention instead to minimising the single model output presented to it.

16.1.4 Preparing for a CMAES_P Run

The Input Dataset

As stated above, the input dataset for CMAES_P is identical to that of PEST. Hence utility software employed for automatic construction of a PEST input dataset can also be employed for construction of a CMAES_P input dataset. Certain features of this dataset, however, are ignored (e.g. variables which govern the calculation of finite–difference derivatives, for the CMA-ES algorithm does not employ these derivatives). Control variables required by the CMA-ES algorithm are solicited from the user upon commencement of CMAES_P execution.

The CMAES_P input dataset can include most of the features available through a PEST input dataset, including the following.

1. Not all parameters cited in the PEST control file need be adjustable, for some parameters can be fixed or tied. In the latter case, the parent parameter to one or more tied parameters is adjusted while the tied parameters simply "ride on its back".

2. Parameters can be log-transformed or untransformed. In the former case the CMA-ES algorithm actually estimates the logs of pertinent parameters. The same rationale that underpins selection of the log transformation option when using PEST should underpin its selection when using CMAES_P.
3. Parameters can be assigned a non-unity SCALE and/or a non-zero OFFSET.
4. Prior information can be included in the PEST control file.
5. The model run by CMAES_P can be a single executable, or a batch/script file comprised of many executables.
6. A covariance matrix can be employed instead of weights for user-specified groups of observations.

Nevertheless, there are some restrictions on a CMAES_P input dataset which do not apply to a PEST input dataset. These include the following.

1. The PESTMODE variable in the "control data" section of the PEST control file must be set to "estimation". It must not be set to "prediction", "regularisation" or "pareto". See the discussion below, however, for a way in which CMAES_P can undertake "pseudo-regularisation".
2. If the NUMCOM variable in the "control data" section of a PEST control file is set to greater than 1, and therefore multiple commands appear in the "model command line" section of the PEST control file. CMAES_P uses only the first of these to run the model.
3. A non-zero setting of MESSFILE is ignored. Thus CMAES_P does not write a PEST-to-model-message file.
4. A non-zero setting of JACFILE is ignored, as are the contents of the "derivatives command line" section of a PEST control file.

Parameter Transformation

As stated above, if parameters are designated as log-transformed during the optimisation process, the CMA-ES algorithm only "sees" the log of these parameters. Thus the internal parameter covariance matrix used in the generation of samples of parameter space will in fact pertain to the logs of pertinent parameters.

As is discussed earlier in this manual, many problems are more linear (and hence more easily solved) if formulated in terms of the logs of parameters, rather than in terms of native parameters. Log transformation also has the ability to make parameter sensitivities more equal, and hence reduce anisotropy of the parameter covariance matrix. However use of an appropriate parameter SCALE can have the same affect, and can be more appropriate if log transformation introduces a significant amount of nonlinearity. Note that a negative SCALE and/or an appropriate OFFSET can maintain positivity in a parameter prior to its log-transformation if the latter is deemed to be useful in certain optimisation contexts.

Parameter Initial Values and Parameter Bounds

Upper and lower bounds are placed on parameter values in the "parameter data" section of the PEST control file; these bounds are respected by CMAES_P as it varies parameters to minimise the objective function.

In normal PEST usage, parameter bounds serve a number of roles. On the one hand they can

provide a "reality check" on parameters estimated through the inversion process. On the other hand they can prevent model instability or other errors that attend the transgression of model-imposed parameter limits.

When using CMAES_P, bounds serve another important role, of which the user should be aware. *Initial parameter standard deviations are computed as one fourth of the interval between user-supplied upper and lower bounds.* As a result of this, random parameter realisations generated at the start of the optimisation process potentially occupy the entirety of the bounded interval. However there will be a tendency for these to be closer to initial parameter values than to parameter bounds, as the former are provided to the CMA-ES algorithm as initial parameter mean values, and hence values of highest likelihood from the random parameter generation point of view.

In recognition of the important role played by bounds and initial values, the following rules should be followed if possible.

1. Bounds should NOT be placed at "plus and minus infinity" (using very high and low numbers) as is sometimes done when using PEST.
2. If appropriate, bounds should be symmetrical about parameter initial values. If parameters are log-transformed, then the logarithms of parameter bounds should be symmetrical about the logarithms of initial parameter values.

It is important to note, however, that initial parameter standard deviations computed in this manner can be overridden through supplying an initial parameter covariance matrix – a matter that is discussed in more detail below.

16.1.5 Running CMAES_P

The CMAES_P Command Line

To inspect CMAES_P command line options, simply type its name at the screen prompt. CMAES_P will respond by writing the following text to the screen.

```
CMAES_P is run using the command:

    cmaes_p pestfile [/p] [/r]

where

    pestfile is the name of a PEST control file,
    /p       is an optional parallelisation switch, and
    /r       is an optional restart switch.
```

As was discussed above, the input dataset for CMAES_P is identical to that of PEST, including a run management file if model runs are to be parallelised. However the serial and parallel versions of CMAES_P employ the same executable; unlike PEST there is no separate "Parallel CMAES_P" executable program. This is further discussed below.

As the optimisation process progresses, CMAES_P records the current value of the objective function to the screen. If model runs are undertaken in serial, then CMAES_P screen output is interspersed with that of the model as both CMAES_P and the model share the same window. In this case it may be convenient to redirect model output to a "null file", named "*nul*" on a WINDOWS platform and "*/dev/null*" on a UNIX platform.

CMAES_P Prompts

After having read the PEST input dataset, CMAES_P presents the user with a series of prompts. But first, if it has established that

1. there is only one observation cited in the PEST control file, and
2. there is no prior information cited in this file,

it asks

```
PEST control file has only one observation.
Minimise model equiv or match to observation? [i/a] (<Enter> if "i"):
```

Enter "i" (or press the <Enter> key) to inform CMAES_P that it must simply lower the value of the single model output corresponding to the single observation cited in the PEST control file. In this case the actual value of that observation, and its weight, listed in the PEST control file is immaterial. On the other hand, enter "a" to instruct CMAES_P to minimise the square of the difference between the single model output and its "observed value" as recorded in the PEST control file. ("i" stands for "minimise" and "a" stands for "match.)

In all other cases of CMAES_P usage, CMAES_P minimises exactly the same weighted least squares objective function as that which PEST does.

Next CMAES_P issues a number of sets of prompts. (Prompts are used rather than an input file as this allows you to run CMAES_P without actually reading this manual; it also reminds you of options available in using CMAES_P. Nevertheless the use of prompts can be a little cumbersome at times.) All prompts include a default response; this default is accepted simply by pressing the <Enter> key.

The first five prompts issued by CMAES_P are listed below; note that the default values provided below pertain to a specific case.

```
Enter values for following CMA control variables:-

   Population size, lambda (<Enter> if 11):
   Number of parents, mu (<Enter> if 6):
   Recombination weights [Superlin, Linear, Equal] (<Enter> if superlin):
   Random number seed (<Enter> if 1111):
   Read parameter covariance matrix from a file? (<Enter> if "n"):
```

The "population size" and "number of parents" are λ and μ respectively discussed in the short theoretical overview of the CMA-ES algorithm provided above. In accordance with suggestions provided in CMA-ES documentation, the default value for the former is computed as

$$\lambda = 4 + 3\ln(n) \qquad (16.1.1)$$

where n is the number of adjustable parameters. The default μ is computed as

$$\mu = \lambda/2 \qquad (16.1.2)$$

After λ model runs have been carried out, a new set of mean parameter values is computed which, in conjunction with the evolving covariance matrix, is employed for generation of new parameter value realisations. The mean is computed through weighting the parameter sets corresponding to the μ best results. If the "equal" alternative is selected in response to the third of the above prompts, then the weights applied to all of the μ parameter sets employed in computation of this mean are equal. If the "linear" option is selected, weights are computed as (after ordering from lowest to highest objective function)

$$w_i = \mu + 1 - i \qquad (16.1.3a)$$

If the "superlin" option is selected, lower objective functions are assigned an even greater weight. Weights are then computed as

$$w_i = \ln(\mu+1) - \ln(i) \qquad (16.1.3b)$$

Any integer can be selected as the random number seed, and supplied to CMAES_P in response to the fourth of the above prompts. Note that the outcome of CMAES_P runs implemented on the same platform will be identical as long as the same random number seed is selected (even if run stopping and restarting is implemented). Selection of different random number seeds results in the internal generation of different parameter set realisations and hence in the following of different optimisation trajectories.

The parameter covariance matrix option (which pertains to the fifth of the above prompts) is discussed in the next subsection.

Next CMAES_P asks (if the number of observations from which the objective function is computed exceeds the number of adjustable parameters)

```
Employ SVD hybridisation?  [y/n] (<Enter> if "n"):
```

The hybridisation scheme is further explained below. At the time of writing, the default is set to "no". However you are urged to try this scheme at some stage, especially if you suspect that derivatives of model outputs with respect to adjustable parameters are not too bad. If a response of "y" is provided to this question, CMAES_P next asks two further questions:

```
How many trial singular value thresholds? (<Enter> if 3):
Use "soft" or "hard" hybridization?  [s/h] (<Enter> if "s"):
```

These are further explained below.

CMAES_P next asks:

```
Forgive model run failure? [y/n]: (<Enter> if "y"):
```

If a response of "n" is provided to this prompt, then a failure on the part of CMAES_P to read any part of any model output file after completion of any model run will precipitate cessation of CMAES_P execution with an appropriate error message. However if the response to the above prompt is "y", CMAES_P will interpret an error in reading model output files as evidence of a model-indigestible parameter set. Internally, it will attribute a very high objective function to this parameter set, thus providing a disincentive to the optimisation process from generating a similar set of parameters.

CMAES_P's next seven prompts are as follows:

```
Termination Criteria:-
   Min rel obj fn change over N itns (<Enter> if 1.00000E-03):
   No of itns (N) over which this applies (<Enter> if 40):
   Min rel param change over N itns (<Enter> if 1.00000E-03):
   No of itns (N) over which this applies (<Enter> if 40):
   Rel high-low generated obj fn diff over N itns (<Enter> if 1.0000E-02):
   No of itns (N) over which this applies (<Enter> if 10):
   Maximum number of iterations (<Enter> if 1000):
```

A number of CMAES_P termination criteria are similar to those employed by PEST. However the default values of variables governing those criteria are different from those commonly employed by PEST.

The first two of the above prompts pertain to the objective function. If it fails to fall by a

relative amount provided in response to the first of the above prompts over the number of iterations provided in response to the second of the above prompts, then the optimisation process will be deemed to have reached completion.

The third and fourth of the above prompts pertain to parameter changes. If no parameter undergoes a relative change given by the response to the third of the above prompts over a number of iterations provided in response to the fourth of the above prompts, then the optimisation process will be deemed to have reached completion.

The fifth and sixth of the above prompts pertain to objective functions computed during any iteration on the basis of the λ trial parameter sets generated during that iteration. If the relative difference between the highest and lowest of these objective functions is small over a number of successive iterations, this indicates that the covariance matrix used to generate parameter realisations contains small parameter variances (hopefully as an outcome of convergence to the global objective function minimum). As time progress, it becomes more and more unlikely that even a small lowering of the objective function will occur under these conditions.

The maximum number of iterations is provided in response to the seventh of the above prompts. If this is supplied as zero, then CMAES_P will undertake just one model run, this being on the basis of initial parameter values as supplied in the PEST control file. This can be useful when the PARREP utility (see part II of this manual) is employed to create a new PEST control file on the basis of values optimised from a previous CMAES_P or PEST run. A single model run can then be undertaken on the basis of these optimised parameter values.

As soon as one of its termination criteria is satisfied, CMAES_P ceases execution, informing you why it has decided to do this.

CMAES_P's final prompt is

```
Run model with initial parameters? [y/n]  (<Enter> if "y"):
```

While the default response to the above prompt is "y", you may prefer to answer "n" in circumstances where model run times are long, especially if CMAES_P undertakes model runs in parallel. In the latter case all machines but one stand idle until this first model run is completed. There is no real advantage to undertaking this run except for the fact that it then "sets the standard" with respect to which objective function outcomes of model runs performed under the control of CMAES_P can be measured. Improvements in the objective function achieved through the calibration process are thereby readily apparent.

Stopping and Restarting

Like PEST, CMAES_P can be stopped by issuing the PSTOP or PSTOPST command from another command line window, opened in the same working folder as CMAES_P. Cessation of execution is instantaneous if CMAES_P is run in parallel mode; however orphaned model runs will continue in their own windows unless halted by the user. When undertaking runs in serial, cessation of CMAES_P execution takes place upon completion of the current model run.

When operating in parallel mode, slaves will not automatically shut down if CMAES_P is halted using the PSTOP command (for often a user-instigated stop is followed by a user-instigated re-commencement of the optimisation process). In contrast, if CMAES_P stops of its own accord because a convergence criterion has been met, of if execution is halted using the PSTOPST command, then slaves automatically shut down.

A stopped CMAES run can be re-started using the "/r" command line switch. Execution re-commences at the beginning of the optimisation iteration at which its execution was previously interrupted. The "/j", "/s" and "/d" restart switches available with PEST are not available with CMAES_P.

If the PPAUSE command is issued from another window, CMAES_P execution will pause at completion of the next model run. This gives you the opportunity to inspect model output files based on current parameters if you so desire. Program execution can be resumed using PUNPAUSE.

Final Model Run

If CMAES_P ceases execution through fulfilment of one of its termination criteria, or if its execution is halted through typing of the PSTOPST command, it undertakes one final model run on the basis of optimised parameters. Thus all model input files are "primed" with these optimised parameter values, and all model output files contain model-generated quantities computed on the basis of these parameter values.

It is important to note, however, that this functionality is not available when CMAES_P is run in parallel mode and stopped using the PSTOPST command. The reason for this is that while cessation of CMAES_P execution may follow rapidly from invocation of the PSTOPST command, current model runs must proceed to completion unless terminated by the user. Once this is done, a final model run on the basis of optimised parameters can be undertaken using the PARREP utility, and then running CMAES_P with the maximum number of iterations set to zero, as described above.

16.1.6 Supplying a Parameter Covariance Matrix

If desired, an initial parameter covariance matrix can be supplied to the CMA-ES algorithm embodied in CMAES_P. This can prove useful for at least the following reasons.

1. A user may desire that initial parameter standard deviations NOT be calculated as one quarter of lower-to-upper parameter bound intervals.
2. When a CMAES_P run is commenced after a PEST run, the parameter covariance matrix computed by PEST or PREDUNC7 can be supplied as the initial parameter covariance matrix for CMAES_P, thus hopefully saving the many model runs required to evolve a suitable covariance matrix.
3. When CMAES_P is run after a previous CMAES_P run, the parameter covariance matrix computed by CMAES_P in the previous process can be supplied as the initial parameter covariance matrix for the next CMAES_P run. Once again, valuable model runs required for covariance matrix evolution can be saved through this strategy.

Upon commencement of execution, CMAES_P's fifth prompt is

```
Read parameter covariance matrix from a file? (<Enter> if "n"):
```

If a user responds to this prompt with "y", CMAES_P then asks

```
Enter name of parameter uncertainty file:
```

The format of a "parameter uncertainty file" is described in part II of this manual. This format provides considerable flexibility in provision of a covariance matrix. In particular

1. Parameters can be divided into independent groups, and a partial covariance matrix can be supplied for each group. If desired, a covariance multiplier can be provided for each such matrix, this allowing easy adjustment of the magnitude of each.
2. Where some or all parameters are statistically independent of other parameters, individual parameter standard deviations can be supplied.

It is important to note that where a parameter is log-transformed, the standard deviation and/or covariance matrix elements associated with that parameter must pertain to the log (to base 10) of its values rather than to native parameter values.

A parameter uncertainty file, and matrices cited therein, can be built in a number of ways. As will be discussed shortly, CMAES_P writes a pair of such files itself at the end of its optimisation process (or part way through this process if it is prematurely halted by the user). Alternatively, if a CMAES_P run was preceded by an "estimation" mode PEST run, the PEST-computed post-calibration parameter covariance matrix may be supplied as an initial covariance matrix for the use of CMAES_P. However a little cutting and pasting will be required for proper formatting of this matrix. Another option is to use the PREDUNC7 utility for construction of a parameter uncertainty file following a previous PEST run.

16.1.7 SVD-Hybridization

Suppose that m parameters are being estimated. After m+1 model runs have been completed (and at intervals of approximately m+1 model runs thereafter), CMAES_P has enough information at its disposal to estimate values for an optimum parameter set using a Gauss method. This estimate would be correct if the model were linear, and if its outputs were uncontaminated by numerical noise. Where these conditions are seriously violated, such an estimate may not be good at all.

CMAES_P actually accomplishes this gradient-based estimation of optimal parameters using singular value decomposition (SVD). Use of singular value decomposition as a parameter estimation device allows CMAES_P to make a number of such estimates, with these estimates varying according to the number of singular values used in their computation. Where nonlinearity or model numerical noise is high, or where parameter differences among the set of m +1 model runs employed in this calculation do not properly span the full dimensionality of parameter space, use of a high number of singular values will probably yield poor estimates of optimal parameters; in contrast, the use of a low number of singular values may not provide enough "parameter resolution" to advance the parameter estimation process, particularly where the optimal parameter set lies within a long narrow valley in parameter space.

In implementing SVD-hybridized parameter estimation, CMAES_P first selects, from among the last m +1 model runs, the parameter set that led to the lowest objective function. It then adopts as "temporary parameters" unit lengths in parameter space along each of the vectors joining this point to the other m points in parameter space that formed the basis of the other m model runs. For each such parameter it then computes an approximation to the derivative of every model output with respect to this parameter as the difference between the respective model output and that for the base model run (this corresponding to the minimum objective function of the m + 1 model runs). Thus it builds a Jacobian matrix **J** for these parameters.

Next CMAES_P undertakes singular value decomposition of $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ (where **Q** is the observation weight matrix). This provides matrices **V** and **E**, calculated as

$$\mathbf{J}^t\mathbf{Q}\mathbf{J} = \mathbf{V}\mathbf{E}\mathbf{V}^t \qquad (16.1.4)$$

where $\mathbf{V}$ is the matrix of normalized eigenvectors of $\mathbf{J}^t\mathbf{Q}\mathbf{J}$, and $\mathbf{E}$ is a diagonal matrix comprised of singular values of $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ arranged in descending order. A subset of these values comprising a matrix $\mathbf{E}_1$ is then selected, and a parameter upgrade vector computed as

$$\mathbf{p} = \mathbf{V}_1\mathbf{E}^{-1}{}_1\mathbf{V}^t{}_1\mathbf{J}^t\mathbf{Q}\mathbf{r} \qquad (16.1.5)$$

where $\mathbf{r}$ is the vector of model-to-measurement residuals computed on the basis of the base parameter set, and $\mathbf{V}_1$ is the subset of $\mathbf{V}$ comprised of eigenvectors corresponding to the subset $\mathbf{E}_1$ of $\mathbf{E}$. Actual parameter values are then computed from this temporary parameter upgrade vector. The parameter upgrade vector is shortened if necessary in order to keep actual parameter values within user-prescribed bounds.

The point at which singular value truncation takes place is determined by singular value ratios. Cut-off ratios of 0.1, 0.01, 0.001 etc. (in that order) are selected, depending on the user's choice of number of truncation levels to test. For each such selection a new parameter set is estimated (unless use of two different singular value ratios results in the same number of selected singular values). The objective function corresponding to each new parameter estimate is then computed either immediately (if serial CMAES_P is being used), or on the next occasion that a package of parallel runs is undertaken (if CMAES_P has been asked to parallelise model runs).

CMAES_P provides two options for assimilating the results of SVD-based parameter upgrade computation into the overall CMA-ES optimisation process, these being labelled as "soft" and "hard" in its prompts. If the "soft" option is taken, the best of the currently-selected λ parameters (these forming part of the m +1 member parameter set on which singular value decomposition analysis was based) is replaced by the SVD-computed parameter set only if the objective function achieved through the singular value decomposition process yields the lowest objective function achieved to date. If the "hard" option is selected, parameter set replacement is undertaken if the SVD-computed parameter set leads to a lower objective function than that computed only on the basis of the current λ parameter sets. The former option is likely to be better if the objective function surface is pitted with many local minima, for the latter option may lead to too rapid convergence and hence heighten the chances of being trapped in a local optimum. However the latter option may lead to more rapid convergence if the objective function surface is relatively free of local optima.

The "cost" of undertaking SVD-hybridisation is that k extra model runs need to be carried out on every second or third CMAES optimisation iteration (depending on the size of λ relative to nm where k is the number of tested singular value truncation levels. If parallelisation of model runs is being undertaken, this may or may not be an issue. Ideally λ should be a multiple of the number of slaves that a user has at his/her disposal in normal CMAES_P operation. When SVD-hybridisation is being employed, $\lambda + k$ should be a multiple of the number of slaves. If there are slaves to spare, then there is no cost whatsoever involved in using SVD-hybridisation, even if no SVD-based attempt at computation of a parameter upgrade yields a better result than that obtained on the basis of the CMA-ES algorithm alone. In other cases, the worth or otherwise of employing SVD-hybridisation will depend on its success in accelerating the optimisation process. In some circumstances the average extra cost per iteration will be more than balanced by the reduced number of iterations required to achieve convergence.

A hidden cost of using the SVD-hybridisation scheme is that CMAES_P memory

requirements can be large. This is because parameters, and all model outputs, associated with m +1 model runs, must be stored in memory. In many cases this is not a high cost; however if many parameters and/or observations are involved in the parameter estimation process, the cost may become high.

At the time of writing, SVD-hybridisation is not allowed if an observation covariance matrix is supplied instead of measurement weights for one or more observation groups.

16.1.8 CMAES_P Output Files

CMAES_P records the history of its optimisation process both to the screen and to its run record file. As already stated, the latter has the same filename base as the PEST control file, but possesses an extension of "*.rec*". Hence if the PEST control file is named *case.pst*, then the run record file is named *case.rec*. This file finishes with a table of optimised parameter values.

At any stage of the optimisation process, best parameters achieved to date are available in a "parameter value file". This is named *case.par*; see section 5.3.2 for specifications of this type of file. Parameter values recorded in this file can be placed into another PEST control file (as initial values listed in that file) using the PEST PARREP utility. At the end of a CMAES_P run, this procedure can be used to create an input dataset for PEST; PEST can then be used to create a series of its own output files based on optimised parameters. To obtain a residuals file set NOPTMAX to zero in the PEST control file so that PEST undertakes just one model run. To obtain parameter covariance and uncertainty data, run PEST in "estimation" model with NOPTMAX set to -1.

CMAES_P records its final covariance matrix in a parameter uncertainty file. This file has the same filename base as the PEST control file but possesses an extension of "*.unc*". Hence it is named *case.unc*. The actual matrix is stored in file *case.cmf*; see part II of this manual for specifications of this type of file.

Using the parameter value and covariance files recorded by CMAES_P, it is possible to commence a new CMAES_P run with new control variables (for example new values for λ and μ) while retaining the benefits of progress made during a previous CMAES_P run. A suitable procedure is as follows.

1. Use the PARREP utility to build a new PEST control file from the PEST control file on which the just-completed CMAES_P run was based, but with initial parameter values replaced by values computed by CMAES_P.
2. Initiate a new CMAES_P run on the basis of this new PEST control file. When CMAES_P prompts for inputs, provide non-default values as desired for variables such as λ and μ. Also inform CMAES_P that a parameter covariance matrix is to be provided; then provide the name of the parameter uncertainty file written by CMAES_P on its previous run.

If appropriate, parameter covariance data can be modified from that recorded in the parameter uncertainty file by CMAES_P on its previous run. Desired modifications may include the following.

1. The variance multiplier recorded in the CMAES_P-produced parameter uncertainty file may be made larger or smaller in order to widen or narrow the initial search region in the re-commenced CMAES_P run.

2. Instead of a covariance matrix, it may be desired that only parameter standard deviations be supplied, these being comprised of the square roots of diagonal elements of the covariance matrix; thus a temporary assumption of parameter independence is made. This can be achieved through use of the PEST MATDIAG utility (see part II of this manual), together with a little cutting and pasting.

16.1.9 Parallelisation

As for PEST, CMAES_P-invoked model runs can be undertaken in parallel across computer networks and/or processors. Model runs are carried out by the PSLAVE program which receive and send signals to CMAES_P through message files. See section 11.2 of this manual for further details.

When parallelisation is implemented, the CMAES_P input dataset must include an additional file, this being the parallel run management file. This should have the same filename base as the PEST control file, but possess an extension of "*.rmf*". This file is identical to the run management file used by Parallel PEST; all variables have the same role, except for the PARLAM variable which, if present, is ignored. It is also important to note that the IFLETYP variable must be set to 0 rather than 1. Thus individual model input and output filenames are not supplied for each slave; rather their names are determined through prefixing the name of the pertinent slave working folder to model input and output filenames supplied in the PEST control file.

As for PEST, CMAES_P records a complete history of communications between it and its slaves to a run management record file; this file is named *case.rmr*.

The number of runs completed during a package of parallel run calls (which normally coincides with an iteration, the total size of the package therefore being λ) is available in a file named *case.rcr*.

There is one major difference between the parallel version of PEST and the parallel version of CMAES_P. Parallel PEST is run as a separate executable (named PPEST). Parallelisation of CMAES_P is activated through a command line switch, this being the "/p" switch. The same executable is employed for both serial and parallel runs. (However the slave PSLAVE is the same as for Parallel PEST.)

The choice to parallelise model runs may have a bearing on the choice of certain CMAES_P control variables. If the parameter population size (λ) is a multiple of the number of slaves, then slaves are not idle for part of an iteration. As has already been discussed, if SVD-hybridisation is implemented, the number of slaves should be a multiple of the population size plus number of singular value cut-offs implemented.

16.1.10 Pseudo-Regularisation

When run in regularisation mode, PEST adjusts weights on an iteration-by-iteration basis for observations and prior information equations belonging to observation groups whose names begin with "regul". Regularisation weights adjustment can take place at two levels.

1. If the IREGADJ regularisation control variable is set to a value other than zero, then weights adjustment between regularisation groups (or even between individual regularisation observations) is undertaken, such that groups (observations) which are most sensitive to parameters which are least sensitive to actual measurements receive the greatest weight.

2. Subsequent to this adjustment, a global regularisation weights multiplier is applied, this being roughly equivalent to the Lagrange multiplier employed in the constrained minimisation process through which Tikhonov regularisation is implemented in PEST. In this constrained minimisation process, the regularisation objective function is minimised subject to the constraint that the measurement objective function is equal to its user-specified target, PHIMLIM.

Such functionality is not available through CMAES_P, for implementation of both of the above weights adjustment strategies requires that sensitivities of model outputs with respect to adjustable parameters be available. CMAES_P does not compute these sensitivities. As has already been discussed, one of its strengths is that it can thus perform satisfactorily in a model calibration context in which model outputs are corrupted by numerical noise. However, as was also discussed above, this comes at a cost. One cost is the fact that model run requirements for objective function minimisation are normally higher than for PEST. Another cost is that implementation of Tikhonov regularisation in the manner discussed above is not possible.

For CMAES_P usage in model calibration contexts this may not be a disadvantage, for normally the number of adjustable parameters will be far fewer than estimated by PEST because of the much higher model run requirements of CMAES_P than of PEST; hence the need for regularisation will not be as pressing. However in contexts where a management objective function is minimised in order to maximise operational efficiency (i.e. in contexts where CMAES_P is being used for optimisation rather than for calibration purposes), the ability to include constraints in the objective function minimisation process may be missed. Certainly the relationships through which constraints are formulated may be included in the CMAES_P objective function. But the ability to compute a weight factor for these constraints equivalent to a Lagrange multiplier is not available. The following procedure may then be worthy of consideration.

1. Set up a PEST control file in which PEST runs in "regularisation" mode. Set NOPTMAX to 1 in this file, so that PEST runs for just 1 optimisation iteration.
2. Run PEST.
3. Build a new PEST control file. In this file replace weights for all observations and prior information equations belonging to regularisation groups with those recorded in the residuals file on the previous one-iteration PEST run. This file has the same filename base as the previous PEST control file but has an extension of "*.res*" (or "*.rei*" for the intermediate residuals file). These weights have been updated by PEST in accordance with the weights adjustment procedures described above.
4. Instruct PEST that it must run in "parameter estimation" mode.

The new PEST input dataset is then ready to be used by CMAES_P.

Of course the question must be asked "why not use PEST to undertake optimisation – after all, we are attaching credence to its derivatives for the purpose of weights adjustment; why not attach credence to these derivatives for the purpose of constrained minimisation of the regularisation objective function?" This is a good question.

16.2 SCEUA_P

16.2.1 Introduction

Like CMAES_P, SCEUA_P is a global optimiser. Like CMAES_P it is interchangeable with PEST in that a PEST input dataset (comprising a PEST control file and template and instruction files cited therein) can also serve as a SCEUA_P input dataset, with no alterations whatsoever required to this dataset. Like CMAES_P, SCEUA_P can work in serial and parallel modes; when used in the latter mode, model runs can be distributed across machines in a network for increased optimisation efficiency. And like CMAES_P, SCEUA_P can be employed to minimise either a least-squares objective function as defined through a PEST input dataset, or an objective function defined in another way by the model.

SCEUA_P implements the SCE ("shuffled complex evolution") algorithm described by Duan (1991) and Duan et al (1992; 1993; 1994); UA stands for "University of Arizona". The SCE method does not require computation of derivatives of model outputs with respect to adjustable parameters. Hence it can operate successfully even where the relationship between model parameters and model outputs is discontinuous, or is contaminated by model-generated numerical noise. It can also perform well where the objective function features local optima on either a large or small scale (or both) in parameter space.

The SCE algorithm was developed primarily for calibration of surface water models, where local optima are not an uncommon occurrence (though it has since been used in many other applications as well). The reader's attention is drawn to the TSPROC program supplied with the PEST Surface Water Utility suite which was also developed specifically for use in the surface water modelling context. This automates preparation of a PEST input dataset (and hence an SCEUA_P input dataset) involving a complex multi-component objective function comprising flows, flow volumes, flow statistics and baseflow-filtered flows.

Code for SCEUA_P encapsulates FORTRAN source code supplied by the University of Arizona. However, as detailed below, the algorithm implemented in that code has been provided with expanded capabilities in its SCEUA_P implementation described herein. It is my hope that no bugs were introduced in the enhancement process. If, for a particular application, a user finds that the SCE algorithm as implemented by SCEUA_P does not perform to expectations, then it is not impossible that this is my fault.

16.2.2 Using SCEUA_P

SCEUA_P Control Variables

The SCEUA algorithm is described in the abovementioned texts; see, in particular, Duan et al (1992). As for most optimisation algorithms, the method can be "tuned" for optimal performance through adjustment of a number of control variables. Default values are supplied for all of these but 1, this being the number of complexes that are evolved in each evolution step undertaken by the algorithm. Choice of fewer complexes requires the carrying out of fewer model runs; however it raises the chances of failure of the SCE process in finding the global objective function minimum. A value of between 2 and 20 is recommended, with the number increasing with complexity of the problem and the number of parameters being estimated. (Use 5 if in doubt.)

As is discussed previously in this manual, log-transformation of some or all parameters can make many optimisation problems easier to solve. Parameter transformation is implemented

through appropriate setting of the PARTRANS variable in the "parameter data" section of the PEST control file. SCEUA_P respects this setting. It also respects the tying and fixing of parameters as implemented in a PEST control file, as well as SCALEs and OFFSETs provided to parameters through this file.

Values for SCEUA_P control variables are provided by the user in response to a series of screen prompts issued by SCEUA_P. This eliminates the need for preparation of a dedicated SCEUA_P input file. On most occasions of SCEUA_P usage, the value of only one control variable needs to be supplied, this being the number of complexes to employ in the current optimisation process. Through responding to all other prompts simply by pressing the <Enter> key, default values are accepted. In most cases these work well.

Running SCEUA_P

SCEUA_P is run by typing its name at the command line prompt, followed by the name of the PEST control file which it must read. Optional parallelisation and/or restart switches can follow that. Thus, for example, if the name of the PEST control file for the current case is *case.pst*, then SCEUA_P should be run by typing the following line at the command line prompt

```
SCEUA_P case
```

Note that either the full name of the PEST control file (i.e. *case.pst* in the example above), or simply its filename base (i.e. *case* in the example above) can be provided on the command line. An optional "/p" switch following the name of the PEST control file signifies parallelisation (see below), while an "/r" switch instructs SCEUA_P to re-start a previous run whose execution was terminated before completion. (Another command line switch, "/s", can also be employed; this is discussed below in conjunction with SCEUA_P parallelisation.)

Immediately upon commencement of execution, SCEUA_P reads the cited PEST control file. If an error exists within this file, the nature of the error is reported to the screen prior to termination of SCEUA_P execution. It is important however that, just as for PEST, the complete input dataset be checked using the PESTCHEK utility (see part II of this manual) prior to running SCEUA_P, for SCEUA_P error checking is not as exhaustive as that of PESTCHEK; certain undetected errors can, in some circumstances, lead to unpredictable SCEUA_P behaviour.

After having read the PEST input dataset, SCEUA_P issues the following series of prompts for values of its control variables. If desired, all but the first can be responded to simply by pressing the <Enter> key, this signalling acceptance of the displayed default value. Alternatively, the user can provide an appropriate value of his/her choice. However if an inappropriate value is supplied, SCEUA_P will respond to the user's input with contempt, repeating the prompt until the user responds with a control variable value that SCEUA_P finds less repugnant.

SCEUA_P's first set of prompts are as follows (note that the default values depicted below are context-specific).

```
Enter initial number of complexes:
Minimum number of complexes (<Enter> if 5):
Parameter sets in each complex (<Enter> if 9):
Parameter sets per sub-complex (<Enter> if 5):
Include initial parameter set in population?  [y/n] (<Enter> if n):
Evolution steps before shuffling (<Enter> if 9):
Random number seed (<Enter> if 555):
```

```
Verbose or non-verbose SCEUA printout?  [v/n]: (<Enter> if "n"):
```

As has been stated above, a suitable response to the first of these prompts is often 5. The SCE algorithm provides the option of decreasing the number of complexes as the optimisation process progresses (this resulting in fewer model runs per complex evolution loop). If such a decrease is desired, provide a number in response to the second of the above prompts that is less than that provided in response to the first of them.

The second, third and fourth of the above prompts are self-explanatory.

As is described by Duan et al (1992), the SCE algorithm commences its operations by generating a number of sets of random parameter values and then undertaking a model run on the basis of each of these. The number of such sets is equal to the number of complexes times the number of parameter sets in each complex. These sets are then grouped into respective complexes after ordering on the basis of objective function value. Parameter value generation in this initial phase of the SCE process takes place on the basis of a uniform probability distribution extending between the upper and lower bound of each parameter, these bounds being supplied in the PEST control file. The user should ensure that these bounds are set no wider than is necessary. Note that if a parameter is log-transformed, the probability distribution on the basis of which its random values are generated is log-uniform rather than simply uniform.

Depending on your response to the fifth of the above prompts, the "initial parameter set" (this being comprised of initial parameter values as supplied in the PEST control file) may be included in the random parameter collection on which initial complex selection is based.

As is common among global optimisers, random number generation underpins much of the SCE optimisation process. This is both the strength and the weakness of such schemes. On the one hand, judicious use of this parameter selection mechanism minimises the probability of being trapped in a local objective function minimum. On the other hand, because these methods don't "head straight downhill" to the perceived lowest point of the objective function terrain, the optimisation process requires many model runs for its completion (mostly many more model runs than are required by PEST). The optimisation process is repeatable if the same random number generator seed is used in subsequent SCEUA_P runs. However selection of a different seed will result in an alternative optimisation path being taken. Sometimes repetition of the optimisation process using a number of different seeds allows the user to verify that the global objective function minimum has indeed been found.

As is described below, one of SCEUA_P's output files is named *sceout.dat*. This is identical to the file of the same name written by the original SCEUA code as supplied by University of Arizona. This original code provides two different options for the recording of data to this file. For the non-verbose option, only the parameter set pertaining to the best estimate of the global optimum at the end of each shuffling loop is recorded. For the verbose option, the entire sample population is recorded at the end of each shuffling loop. Note, however, that parameters in this file are referred to as "x1", "x2" etc. Depending on the log transformation status of various parameters, the values recorded in this file can be either native parameter values or the logs (to base 10) of these values; the values of tied and fixed parameters are omitted altogether. (As is discussed below, a somewhat more informative, PEST-compatible, record of the optimisation process is available through the SCEUA_P run record file and its associated parameter value file.)

SCEUA_P's next three prompts pertain to termination of the optimisation process. They are as follows:

```
Max relative obj fn change over N itns (<Enter> if 1.0000E-02):
No of itns over which this applies (<Enter> if 5):
Maximum number of model runs (<Enter> if 10000):
```

If the default values are accepted, SCEUA_P will cease execution if the objective function is not reduced by more than one percent over five successive iterations (i.e. evolution loops), or if 10000 model runs have been carried out.

If a value of zero is supplied in response to the last of the above prompts, SCEUA_P will carry out just one model run using the initial parameter values supplied in the PEST control file. It will then terminate execution after writing the objective function value corresponding to this parameter set (as well as contributions to this objective function by different observation groups) to the screen and to its run record file. This can be a useful device for ensuring that model input files contain optimised parameter values and that model output files contain predictions generated on the basis of these optimised parameter values. If a previous SCEUA_P run was halted prematurely (but the user nevertheless has confidence that near-optimal parameter values were obtained), or was undertaken in parallel mode, then the PARREP utility (see part II of this manual) can be employed to build a new PEST control file using parameter values computed on this previous run. By then running SCEUA_P with the "maximum number of model runs" set to zero, the desired aim of building model input and output files on the basis of these values is thereby achieved.

SCEUA_P Output

As it runs, SCEUA_P writes the current value of the objective function (and the contributions made to the objective function by different observation groups) to the screen. This information is replicated in the SCEUA_P run record file. This file is named *case*.*rec*. The values employed for SCEUA_P control variables for the current run are recorded at the top of this file; optimised parameter values are tabulated at the end of the file.

At any stage of the optimisation process, best parameter values achieved to date can be found in a parameter value file named *case*.*par*; see section 5.3.2 for specifications of this type of file. As mentioned above, the PARREP utility can be used to insert these values into a new PEST control file as initial parameter values in that file.

As discussed above, SCEUA_P produces a third file named *sceout.dat*. This contains a complete record of the SCE optimisation process written in SCE format.

Interrupting a SCEUA_P Run

SCEUA_P execution can be stopped in its tracks by pressing <Ctl-C>. It can be stopped in a less brutal way by typing PSTOP while situated in the same folder as that employed for the current SCEUA_P run, but within another command line window. If a final model run based on optimised parameters is desired before cessation of execution, use the PSTOPST command instead (unless SCEUA_P is running in parallel mode and therefore cannot write this file for reasons discussed below). Execution of SCEUA_P can be temporarily paused using the PPAUSE command, and re-commenced using the PUNPAUSE command. Note that if SCEUA_P is being used in serial mode, none of these commands will take effect until completion of the current model run. If running in parallel mode they will take effect immediately, while the model runs to completion in other window(s) and/or on other machine(s).

Restarting a SCEUA_P Run

A prematurely terminated SCEUA_P run can be restarted by adding the "/r" switch to its command line, after the name of the PEST control file which it must read. If execution was halted during the initial phase of SCEUA_P execution prior to the first instance of complex evolution (i.e. at that stage of the process where model runs are simply undertaken on the basis of random parameter values), then re-commencement of SCEUA_P execution will actually take place at the start of the entire SCE optimisation process. Otherwise SCEUA_P will re-commence execution at the beginning of the evolution loop in which its execution was previously interrupted.

Note that in spite of the fact that random number generation forms the basis of the SCE algorithm, a stopped and re-started run will follow the same optimisation path as that which would have been followed if no interruption had taken place.

Minimisation of a Non-Least-Squares Objective Function

If there is no prior information in the PEST control file, and if only one observation is featured in this file, SCEUA_P issues an additional prompt to those listed above. It asks

```
Minimise model equiv or match to observation? [i/a] (<Enter> if "i"):
```

If "i" (for "minimise") is selected, SCEUA_P does not compute an objective function through forming the difference between the single model output and its single measured counterpart and multiplying by the square of the weight. Instead it simply minimises the single model output, which can therefore be an objective function computed by the model in a way that best suits the user's current application. Selection of "a" (for "match") results in normal SCEUA_P operation in which SCEUA_P attempts to minimise the difference between the single model output and its observed counterpart. Note that the lowest objective function achievable through the latter process is zero. The lower limit to a model-computed objective function depends on the way in which this objective function is defined. There is no reason why this cannot be less than zero.

Final Model Run

If the SCE optimisation process runs to completion, or if it is stopped using the PSTOPST command, SCEUA_P will undertake one final model run using optimised parameter values. This does not occur, however, when SCEUA_P runs in parallel mode for reasons which will be discussed below. In this latter case the PARREP utility can be used to build a new PEST control file whose initial parameter values are in fact optimised parameter values. By setting NOPTMAX to zero in this file, SCEUA_P (or PEST for that matter) can then be used to undertake just one model run on the basis of this parameter set.

Best Parameter Values

As has already been discussed, at any time during a SCEUA_P run the best parameter values achieved so far during that run are available in a parameter value file named *case.par*. Thus if SCEUA_P execution is terminated through user intervention or model failure, the best parameter values achieved during the run are nevertheless available for later use. The role of the PARREP utility in facilitating the conduction of a single model run on the basis of these parameters has been described. (Sometimes it may be noticed that use of PARREP in this fashion leads to a lower objective function than that listed in the SCEUA_P run record file. The objective function is reported in the latter file only at the end of each complex evolution

loop. If a lower objective function is encountered mid-loop the improved parameter set is immediately recorded in the parameter value file.)

16.2.3 Parallel SCEUA_P

Principles of Use

Parallelisation of the SCE optimisation process is not as easy as that of PEST, or of the CMA optimisation process. In both of these latter algorithms batches of model runs are intermittently required for which the parameters employed in one such run are independent of those employed in any of the others. Unfortunately this is not the case for the SCE algorithm, for evolution of parameter complexes is an innately serial process in which the parameters that are employed on one run are dependent on the objective function achieved during the previous model run.

Nevertheless, parallelisation is not impossible, for the evolution of complexes is independent and can thus be undertaken for different complexes on different machines. The procedure with which the SCE process begins (in which a suite of model runs is undertaken on the basis of randomly selected parameter values) is also parallelizable.

Parallelisation at the level of parameter complexes has been implemented in SCEUA_P. However while the use of SCEUA_P in parallel mode is similar to that of PEST (in that a master and multiple slaves are employed for this purpose), there are nevertheless some important differences of which the user must be aware. These are now outlined.

The SCEUA_P Master

In contrast to PEST, the program which implements the parallelised version of the SCE algorithm is the same as that which implements the serial version of this algorithm; parallelisation is invoked simply through use of the "/p" switch on the command line. Thus to run SCEUA_P in parallel mode on the basis of a calibration dataset encapsulated in a PEST control file named *case.pst*, SCEUA_P should be run using the command

```
sceua_p case /p
```

The restart "/r" switch can be added to the end of this line if desired.

As for Parallel PEST and CMAES_P, slave details must be supplied to SCEUA_P through a run management file named *case.rmf*. The format of this file is the same as that used by PEST. However the PARLAM variable is ignored, and IFLETYP must be set to zero.

Slaves

As for Parallel PEST and CMAES_P, slave functionality is implemented in the PSLAVE program. However there are a number of important differences between the tasks required of a slave when serving PEST and CMAES_P on the one hand, and SCEUA_P on the other hand. In the former cases the slave supervises the carrying out of a single model run at a time. Hence when a user is prompted for the command which the slave must use to run the model, he/she simply supplies the model command. However when PSLAVE serves a SCEUA_P master, the command to run SCEUA_P in "slave mode" must be supplied to PSLAVE as the "model command line", for the "model" in this case is actually a special version of SCEUA_P which either undertakes a single model run, or evolves one parameter complex, as required during pertinent stages of the SCE optimisation process. Thus the slave runs SCEUA_P; it is SCEUA_P which actually runs the model (many times).

There are a number of important consequences of this mode of slave operation. Some of these are as follows.

1. When prompted by PSLAVE for the command to run the model, you must respond with:-

   `sceua_p case /s`

   where ***case*** is the filename base of the PEST control file – the same file as that supplied on the command line of the SCEUA_P master. "/s" is a switch which commands SCEUA_P to run in slave mode.

2. Because of this, the PEST control file, and all template and instruction files cited therein, must be copied to all slave directories from the master directory. This differs from the operation of Parallel PEST (but not of BEOPEST) wherein the PEST master writes model input files and reads model output files across the network based on template and instruction files residing only in its own working directory.

Other important consequences of the fact that parallelisation is undertaken on the basis of parameter complexes rather than individual model runs include the following.

1. Once the original set of model runs based on random parameter values has been carried out, there is no advantage to having more than *n* slaves, where *n* is the user-supplied number of complexes that SCE has been instructed to use in the optimisation process.
2. On the other hand, if more than *n* slaves are available, there is no advantage in informing SCE that it must use less than *n* complexes.
3. Because the number of model runs required in evolution of different complexes may be different, some slaves may be idle while waiting for the evolution of other complexes to reach completion, prior to the shuffling of parameter sets undertaken by the master program.
4. Because the use of random numbers is an integral component of the SCE optimisation process, the optimisation path taken for a given problem will be different when solved by SCEUA_P in parallel mode from that taken by SCEUA_P in serial mode, even if the same random number generator seed is employed in both cases. This is because the random number generator cannot be used to generate parameter values for the same complexes in the same order in both of these two modes. (Nevertheless a restarted parallel run will indeed follow the same optimisation path as an uninterrupted parallel solution process.)

When run in slave mode, SCEUA_P leaves no output files, for only the master program leaves a record of the optimisation process and the objective function and parameter values achieved through this process. The master SCEUA_P program also works very differently when running in parallel mode than when running in serial mode. In fact it writes its own template file (this being called *sce_slave.tpl*) which in turn governs the writing of the SCEUA_P slave input file named *sce_slave.in*. The SCEUA_P slave in turn writes an output file named *sce_slave.out* which is read by the master SCEUA_P program using an instruction file named *sce_slave.ins* written by itself.

Final Model Run in Parallel Mode

Because of the changes made to its normal operations, the parallelised SCEUA_P master

program cannot undertake a final model run using optimised parameters upon completion of the optimisation process. This must be done using the PARREP utility discussed above.

Additional Output Files

In addition to its normal suite of output files, the SCEUA_P master records an additional file named *case.rcr*, where *case* is the filename base of the PEST control file for the current case. This is updated on every occasion that SCEUA_P requests a model run; *case.rcr* simply records the number of model runs undertaken to date. It must be recalled in interpreting this file, that during the parameter complex evolution process, a "model run" is actually an SCEUA_P run undertaken in slave mode. Parameter complex evolution in fact requires many actual model runs, all of these being undertaken by the SCEUA_P slave.

A parallel run management file named *case.rmr* is also written by the SCEUA_P master when running in parallel mode. This provides a complete record of communication between the SCEUA_P master program and each of its slaves.

A Possible Error Message

As stated above, when undertaking parallelised SCE optimisation, it is actually the slave version of SCEUA_P which runs the model. It does this through a call to the operating system. However the slave version of SCEUA_P is actually run by PSLAVE through an operating system call. Thus the model is actually run by a system call nested within a system call.

Normally this situation presents no problems to an operating system. However it has been noticed that if a computer is very busy with other tasks, an error message such as "cannot read model output file as file sharing is not loaded", may be written to the slave window. This occurs because operating system messages regarding the availability (or lack of availability) of files can get overtaken by other messages to read these files.

If this occurs too often, try setting the value of the RUNREPEAT variable in the run manager file to 1 (see section 11.2 of this manual for a description of this variable). This will force SCEUA_P to repeat any model run in which this (or any other) error occurs before declaring the problem to be the model's fault and terminating the optimisation process with an appropriate error message as a consequence.

16.2.4 Problem Formulation

When using the SCE method, take care to formulate a well posed inverse problem, i.e. a problem for which there is a unique solution. The SCE algorithm has no built-in regularisation; hence it is up to the user to undertake pre-calibration "manual regularisation" by ensuring that an appropriate level of parameter parsimonisation has been undertaken prior to the parameter estimation process, and that not too much post-calibration correlation is likely to exist between parameters that are retained in this process. The performance of SCE (and most other unregularised parameter estimation methods) deteriorates rapidly as the level of post-calibration parameter correlation increases.

17. SENSAN

17.1 Introduction

In many modelling applications, it is useful to examine the sensitivity of particular model outputs to particular model inputs. Such an analysis may be required as part of an effort to increase a modeller's understanding of the processes simulated by the model. Or it may be the first step in a model calibration exercise whereby key system parameters are identified.

SENSAN facilitates the sensitivity analysis process by allowing a modeller to automate the tedious task of adjusting certain model inputs, running the model, reading the outputs of interest, recording their values, and then commencing the whole cycle again. Using SENSAN, a modeller can prepare for an unlimited number of model runs and then let the computer undertake these runs overnight, over a weekend, or simply while he/she is doing other things. SENSAN reads user-prepared parameter values and writes specified model output values to files which can easily be imported to a spreadsheet for further processing.

If requested, a system command can be issued after each model run. For example a user may wish to rename certain model output files after some model runs have been completed; hence these model output files are not overwritten during subsequent model runs and are thus available for later inspection.

SENSAN is model-independent. This means that it can be used to conduct a sensitivity analysis in conjunction with any model. It achieves this by communicating with a model through the model's own input and output files. It uses an identical model interface protocol to that of PEST, writing model input files on the basis of user-supplied templates, and reading model output files with the aid of a user-prepared instruction set. In fact, SENSAN communicates only indirectly with a model. It actually uses the PEST TEMPCHEK and INSCHEK utilities to write and read model files; these programs are run by SENSAN as "system calls". TEMPCHEK and INSCHEK are documented in part II of this manual.

Like PEST, SENSAN runs a model through a command supplied by the user. There is no reason why a "model" cannot be a batch file housing a number of commands. Thus a "model" can consist of a series of executables, the outputs of one constituting the inputs to another, or simply a number of executables which read different input files and generate different output files. SENSAN can write parameter values to many input files and read model outputs from many output files.

SENSAN is limited in the number of parameters and observations that it can handle, both through the internal dimensioning of its own arrays and those belonging to TEMPCHEK and INSCHEK which it runs. This will rarely pose a problem, for it is in the nature of sensitivity analysis that adjustable parameters do not number in the hundreds nor selected model outcomes in the thousands. Nevertheless if either SENSAN, TEMPCHEK or INSCHEK reports that it cannot allocate sufficient memory to commence or continue execution, or that the maximum number of parameters or observations has been exceeded, contact Watermark Numerical Computing for versions of SENSAN, TEMPCHEK and INSCHEK in which these restrictions are lifted.

A comprehensive SENSAN input data checker named SENSCHEK is provided with SENSAN. Its role is similar to that of PESTCHEK and should be run after all SENSAN input data has been prepared, prior to running SENSAN itself. Like PESTCHEK, SENSCHEK is

documented in part II of this model.

You should note that SENSAN is an old program, being one of the original members of the PEST suite. Unlike PEST, it has not been modernized over the years since it was originally written. Nor has it been parallelised. Furthermore, SENSAN's use of TEMPCHEK and INSCHEK to write and read model input and output files can sometimes incur operating system objections if a computer is simultaneously busy with other tasks.

17.2 SENSAN File Requirements

17.2.1 General

SENSAN requires four types of input file. The first two are the SENSAN control file and a so-called "parameter variation file". The former file provides SENSAN with the structural details of a particular sensitivity analysis. The latter provides SENSAN with the parameter values to be used in the succession of model runs which it must undertake. The other two file types are PEST template and instruction files. These latter two kinds of file are dealt with briefly first.

17.2.2 Template Files

Section 2.2 of this manual provides a detailed discussion of how PEST writes parameter values to model input files.

After you have prepared a template file prior to running PEST, you can check its integrity using the PEST utility TEMPCHEK. As explained in part II of this manual, TEMPCHEK also provides the functionality to generate a model input file on the basis of a template file and a corresponding user-supplied list of parameter values. SENSAN runs TEMPCHEK whenever it wishes to prepare a model input file on the basis of a set of parameter values. Thus it is essential that the directory (i.e. folder) in which the executable file *tempchek.exe* resides is either the current working folder or is a folder that is cited in the PATH environment variable.

You can provide SENSAN with the name of a single template file in order that it can generate a single model input file. Alternatively, you can provide SENSAN with the names of many template files in order to generate multiple input files prior to running the model. In either case, before it runs the model, SENSAN writes a parameter value file using the current set of parameter values as provided in the parameter variation file (see below). Then SENSAN runs TEMPCHEK for each model input file which must be produced. It then runs the model.

Before running SENSAN you should always check the integrity of all template files which you supply to it by running TEMPCHEK yourself outside of SENSAN.

17.2.3 Instruction Files

Instruction files are discussed in section 2.3 of this manual. Model-generated numbers can be read from one or many model output files as long as at least one instruction file is provided for each model output file. The integrity of an instruction file can be checked using the PEST utility INSCHEK described in part II of this manual. INSCHEK is also capable of actually reading values from a model output file on the basis of a user-supplied instruction file. SENSAN runs INSCHEK to read model output files after it has run the model. Hence while a user must supply SENSAN with the instruction files required to read one or more model

output files, it is actually INSCHEK which reads these files; SENSAN then reads the "observation value files" written by INSCHEK in order to ascertain current model outcome values. Because SENSAN must run INSCHEK at least once every time it runs the model, it is essential that the executable file *inschek.exe* resides either in the current working folder or within a folder that is cited in the PATH environment variable.

Before running SENSAN, you should check the integrity of all instruction files which you supply to it by running INSCHEK yourself outside of SENSAN.

17.2.4 The Parameter Variation File

SENSAN's task is to run a model as many times as a user requires, providing the model with a user-specified set of parameter values on each occasion. As discussed above, the parameters which are to be varied from model run to model run are identified on one or a number of template files. The values which these parameters must assume on successive model runs are provided to SENSAN in a "parameter variation file", an example of which is presented in figure 17.1.

```
dep1    dep2    res1    res2    res3
1.0     10.0    5.0     2.0     10.0
2.0     10.0    5.0     2.0     10.0
1.0     11.0    5.0     2.0     10.0
1.0     10.0    6.0     2.0     10.0
1.0     10.0    5.0     3.0     10.0
1.0     10.0    5.0     2.0     11.0
```

Figure 17.1 A parameter variation file.

The file shown in figure 17.1 provides 6 sets of values for 5 parameters; the parameter names appear in the top row. As usual, a parameter name must be twelve characters or less in length. The same parameter names must be cited on template files provided to SENSAN. In fact, if there is a naming discrepancy between the parameters cited in the parameter variation file and those cited in the template files supplied to SENSAN, parameters cited in the parameter variation file which are absent from any template file(s) will not be provided with updated values from model run to model run. This will manifest itself on SENSAN output files as a total lack of sensitivity for some parameters named in the parameter variation file. A comprehensive checking program named SENSCHEK (see part II of this manual) has the ability to detect such inconsistencies in the SENSAN input dataset. Hence SENSCHEK should always be run prior to running SENSAN.

The second and subsequent rows of a parameter variation file contain parameter values for SENSAN to use on successive model runs. A separate model run will be undertaken for each such row. A parameter variation file can possess as many rows as a user desires; hence SENSAN can be set up to undertake thousands of model runs if this is considered necessary (as it may be in Monte Carlo simulation).

In many sensitivity analyses, a user is interested in the effect of varying parameters, either individually or in groups, from certain "base" values. In such cases, parameter base values should appear on the second line of the parameter variation file immediately under the parameter names. As will be discussed below, SENSAN produces two output files in which variations from "base value outputs" are recorded, "base value outputs" being defined as model outputs calculated on the basis of base parameter values.

Items on each line of a parameter value file can be space, comma or tab-delimited.

17.2.5 SENSAN Control File

It is recommended, though it is not essential, that the SENSAN control file be provided with a filename extension of "*.sns*". Use of this default extension avoids the need to type in the entire SENSAN control filename when running either SENSAN or SENSCHEK.

Figure 17.2 shows a SENSAN control file. Figure 17.3 provides specifications of the SENSAN control file. As is apparent, the SENSAN control file resembles, to some extent, the PEST control file. Like the PEST control file, the SENSAN control file must begin with a three-character code, namely "scf", identifying it as a SENSAN control file. Like the PEST control file, the SENSAN control file is divided into sections by lines beginning with the "*" character. And like the PEST control file, the SENSAN control file provides information to SENSAN through the values taken by certain input variables, many of which are also used by PEST. Where such variables are, indeed, used by PEST, they have identical meanings for both SENSAN and PEST.

```
scf
* control data
noverbose
5 19
2 3 single point
* sensan files
parvar.dat
out1.dat
out2.dat
out3.dat
* model command line
model > nul
* model input/output
ves.tp1 ves1.inp
ves.tp2 ves2.inp
ves1.ins ves1.out
ves2.ins ves2.out
ves3.ins ves3.out
```

Figure 17.2 A SENSAN control file.

```
scf
* control data
SCREENDISP
NPAR NOBS
NTPLFLE NINSFLE PRECIS DPOINT
* sensan files
VARFLE
ABSFLE
RELFLE
SENSFLE
* model command line
write the command which SENSAN must use to run the model
* model input/output
TEMPFLE INFLE
(one such line for NTPLFLE template files)
INSFLE OUTFLE
(one such line for NINSFLE instruction files)
```

Figure 17.3 Specifications of the SENSAN control file.

The role of each SENSAN input variable is now discussed.

17.2.6 Control Data Section

SCREENDISP

SCREENDISP is a character variable which can take either one of two possible values. These values are “noverbose” and “verbose”. In the former case, when SENSAN runs TEMPCHEK and INSCHEK it redirects all of the screen output from these programs to the null file; hence the user is not aware that they are running. In the latter case, TEMPCHEK and INSCHEK output is directed to the screen in the usual fashion.

Once you have set up a SENSAN run and ensured that everything is working correctly, a nicer screen display is obtained by using the “noverbose” option. In this case you should ensure that the model likewise produces no screen output by redirecting its output to the null file using, for example, the command

```
model > nul
```

to run the model. For a UNIX system this command becomes

```
model > /dev/null
```

If SENSAN is thus left to produce the only screen output, you can monitor its progress and detect any SENSAN error messages if they are written to the screen.

NPAR

This is the number of parameters. It must agree with the number of parameters cited in the template file(s) used by SENSAN. It must also agree with the number of parameters named in the parameter variation file provided to SENSAN.

NOBS

NOBS is the number of “observations”, i.e. the number of model outcomes used in the sensitivity analysis process. It must agree with the number of observations cited in the instruction file(s) provided to SENSAN.

NTPFLE

NTPFLE is the number of template files to be used by SENSAN. For each template file there must be a corresponding model input file; see below. Note that a given template file can be used to write more than one model input file; however two templates cannot write the same model input file.

NINSFLE

NINSFLE is the number of instruction files used by SENSAN to read model outcomes. For each instruction file there must be a matching model output file. Note that the same instruction file cannot read more than one model output file (observation values would be overwritten); however two different instruction files can read the same model output file.

PRECIS

PRECIS is a character variable which must take either the value “single” or “double”. It determines whether single or double precision protocol is used to represent a very large or very small number, or a number in a wide parameter space; see section 2.2.6 for more details. The value “single” is usually appropriate.

DPOINT

DPOINT must be supplied as either "point" or "nopoint". In the latter case the decimal point is omitted if there is a tight squeeze of a parameter value into a parameter space. Use "point" if at all possible, for some models make assumptions regarding the location of a missing decimal point. See section 2.2.6 for more details.

17.2.7 SENSAN Files Section

VARFLE

VARFLE is the name of the parameter variation file for the current model run. The number of parameters cited in this file must agree with the value of NPAR cited in the "control data" section of the SENSAN input file.

ABSFLE, RELFLE and SENSFLE

The names of the three SENSAN output files. Contents of these files are discussed below.

17.2.8 Model Command Line Section

Provide the command that you would normally use to run the model. Remember that you can enter the name of a batch file here to run a model consisting of multiple executables. To prevent screen output from occurring during execution of batch file commands (if desired) you can disable echoing of each batch file command using the "@" character and the "echo off" command. Also, screen output produced by individual model executables can be redirected to the null file. See figure 17.4.

```
@echo off
model1 > nul
model2 > nul
```

Figure 17.4 A batch file serving as a model; all screen output has been disabled.

Care should be taken if SENSAN is executing in "noverbose" mode for it then appends the string "> nul" to the command recorded in the "model command line" section of its input file. If the command already involves output redirection to a file using the ">" symbol, this may become confounded through use of the further ">" symbol supplied by SENSAN to redirect command output to the *nul* file.

17.2.9 Model Input/Output Section

TEMPFLE

TEMPFLE is the name of a template file used to write a model input file.

INFLE

INFLE is the name of a model input file corresponding to the template file preceding it in the SENSAN control file.

INSFLE

INSFLE is the name of a PEST instruction file.

OUTFLE

This is the name of the model output file read by the instruction file whose name precedes it in the SENSAN control file.

17.2.10 Issuing a System Command from within SENSAN

SENSAN allows a user to issue a system command after each model run. A system command is a direction to the operating system, and is implemented by the system just as if the command were typed at the screen prompt. The command can be an operating system command such as "copy" or "del"; or it can be the name of a user-supplied executable program or batch file.

The system command to be run after any particular model run should be written to the parameter variation file following the parameter values pertinent to that model run. In many cases the command will simply be the "copy" command, ensuring that model output files are stored under different names before they are overwritten during subsequent model runs. Figure 17.5 shows such a case.

```
dep1    dep2    res1    res2    res3
1.0     10.0    5.0     2.0     10.0 copy model.out model1.out
2.0     10.0    5.0     2.0     10.0 copy model.out model2.out
1.0     11.0    5.0     2.0     10.0 copy model.out model3.out
1.0     10.0    6.0     2.0     10.0 copy model.out model4.out
1.0     10.0    5.0     3.0     10.0 copy model.out model5.out
1.0     10.0    5.0     2.0     11.0 copy model.out model6.out
```

Figure 17.5 A parameter variation file for a SENSAN run in which system commands are run after the model.

For the example in figure 17.5 the model run by SENSAN produces a file called *model.out*. After each model run, this file is copied to a different file whose name is associated with that run. These files can later be inspected or processed in a fitting manner.

SENSAN assumes that any characters following the NPAR numbers representing the NPAR parameter values to use for a particular model run constitute a system command which it duly delivers to the operating system after the model has run. SENSAN takes no responsibility for incorrect commands; nor does it check whether the system has properly interpreted and executed the command. It simply reads the next set of parameters and undertakes the next model run after control has been returned back to it from the operating system after the latter has been provided with the command.

Care should be taken if SENSAN is executing in "noverbose" mode, for then the string "> nul" (or "> /dev/null" in UNIX) is added to any command appearing in the parameter variation file. This may cause problems if a command already uses the ">" symbol to redirect its output to a file.

17.3 Running SENSAN

17.3.1 SENSAN Command Line

SENSAN is run using the command

`sensan` ***infile***

where ***infile*** is the name of a SENSAN control file. If the latter possesses an extension of

"*.sns*" it is not necessary to include this extension in the SENSAN command line, for SENSAN automatically appends "*.sns*" to a filename supplied without extension.

It is important to ensure before SENSAN is run that the executable files *tempchek.exe* and *inschek.exe* are either in the current folder, or are in a folder that is cited in the PATH environment variable. As is mentioned above, SENSAN runs both of these programs in the course of its execution, the first to generate model input files and the second to read model output files.

17.3.2 Interrupting SENSAN Execution

To interrupt SENSAN type <Ctl-C>.

17.4 Files Written by SENSAN

17.4.1 SENSAN Output Files

SENSAN produces three output files, each of which is easily imported into a spreadsheet for subsequent analysis. In each of these files the first NPAR columns contain the parameter values supplied to SENSAN in the parameter variation file. The subsequent NOBS columns pertain to the NOBS model outcomes (i.e. "observations") cited in the instruction file(s) supplied to SENSAN. The first row of each of these output files contains parameter and observation names.

The last NOBS entries on each line of the first SENSAN output file (ABSFLE) simply list the NOBS model outcomes read from the model output file(s) after the model was run using the parameter values supplied as the first NPAR entries of the same line.

The second SENSAN output file (RELFLE) lists the relative differences between observation values on second and subsequent data lines of the ABSFLE output file and observation values cited on the first data line. Hence if the first data line (i.e. the line following the parameter name line) of the parameter variation file lists parameter base values, the second SENSAN output file lists the variations of model outcome values relative to model outcome base values. If, for a particular model outcome, o_b represents the base value, and o_p represents the value for a certain set of alternative parameter values, then the value written to the RELFLE output file for that model outcome and parameter set is

$$\frac{o_p - o_b}{o_b} \qquad (17.4.1)$$

Note that if o_b is zero, a value of 10^{35} is written to RELFLE as an indicator of this condition.

The third SENSAN output file (SENSFLE) provides model outcome "sensitivities" with respect to parameter variations from their base values. As usual, parameter base values are assumed to reside on the first data line of the parameter variation file. Sensitivity for a particular outcome is calculated as the difference between that model outcome and the pertinent model outcome base value, divided by the difference between the current parameter set and the parameter base values. The latter is calculated as the L_2 norm, i.e. the square root of the sum of squared differences between a current parameter set and the base parameter set. Thus if only a single parameter p differs from the base set, the sensitivity for a particular observation o is defined as

$$\frac{o-o_b}{p-p_b} \qquad (17.4.2)$$

where o_b and p_b are model outcome and parameter base values and o and p are the model outcome and parameter values pertaining to a particular model run. Hence if NPAR+1 parameter sets are provided to SENSAN, where the first set contains parameter base values and the subsequent NPAR sets contain parameter values identical to the base values except that each parameter in turn is varied from the base value by an incremental amount, then the last NPAR rows and NOBS columns on the SENSAN sensitivity output file, SENSFLE, approximates the transpose of a derivatives matrix.

Note that if p-p_b in equation 17.4.2 is equal to zero, then SENSAN writes the corresponding sensitivity as 10^{35}, except for the first data line (assumed to be the base value line) where all sensitivities are provided as 0.0. Note also that the L_2 norm can only be positive. However when only a single parameter is varied, the sign of that variation is taken into account, resulting in a negative denominator for equation 17.4.2 if $p < p_b$.

17.4.2 Other Files used by SENSAN

As has already been discussed, SENSAN uses programs TEMPCHEK and INSCHEK to prepare model input files and read model output files. SENSAN writes a parameter value file for the use of TEMPCHEK, naming this file *t###.par*. This name should be avoided when naming other files.

As explained in part II of this manual, INSCHEK records the values of the observations which it reads from a model output file in the observation value file *instruct.obf* where *instruct* is the filename base of the instruction file provided to INSCHEK. Hence for any instruction file provided to SENSAN, use of a file with the same filename base but with an extension of "*.obf*" will result in that file being overwritten.

17.5 Sensitivity of the Objective Function

SENSAN allows you to undertake many model runs without user intervention. The sensitivity of certain model outputs to certain parameters can be tested. However SENSAN does not compute an objective function because it does not read an observation dataset, and hence cannot compare model outputs with corresponding observations to calculate residuals.

However once all PEST input files have been prepared for a particular case, SENSAN can be used in conjunction with PEST to study the dependence of the objective function (or any components thereof) on certain parameters. Where there are only two parameters, this can be used to contour the objective function in parameter value space.

A SENSAN control file implementing this is shown in figure 17.6.

```
scf
* control data
verbose
6 1
1 1 single point
* sensan files
parvar.dat
out1.txt
out2.txt
out3.txt
* model command line
```

```
pest ves4
* model input/output
pst.tpl ves4.pst
rec.ins ves4.rec
```

Figure 17.6 A SENSAN control file with PEST as the model.

Note the following points.

- In the PEST control file the value of NOPTMAX should be set to zero. Hence PEST runs the model only once before it terminates execution.
- There is only one template file and one instruction file.
- The template file is built from the PEST control file. Parameters adjusted by SENSAN are initial parameter values as listed on the PEST control file.
- SENSAN's observation file is the run record file for the PEST case.
- As in normal SENSAN operation, supply parameter values to be used by SENSAN through a parameter variation file.

The instruction set by which the PEST control file may be read is shown below (the observation name is "phi").

```
pif $
$(ie phi)$ $=$ !phi!
```

This instruction set simply instructs SENSAN to read the PEST run record file until it encounters the string "(ie phi)" followed by "=", and then to read the observation named "phi" as a non-fixed observation following that. More complex instruction sets are required where multiple objective function components must be read.

17.6 SENSAN Error Checking and Run-Time Problems

As has already been discussed, SENSAN does not carry out extensive error checking. Comprehensive SENSAN input data error checking can be undertaken using SENSCHEK. Hence if there are any problems encountered in SENSAN execution, or if there are any suspicions regarding the numbers recorded on any of its output files, SENSCHEK should be run immediately if it has not already been run.

If SENSCHEK has not been used to verify an input dataset and SENSAN finds an error in a parameter variation file (such as an unreadable parameter value) it will not terminate execution. Instead, SENSAN reports the error to the screen and moves on to the next parameter set. However it writes the offending line of the parameter variation file to its three output files. If a trailing system command is present on this line, this too will be written to the SENSAN output files; however the command is not executed. Naturally model outcome values are not written to the SENSAN output files because they cannot be calculated in these circumstances.

It is not impossible that model execution will fail for some parameter value sets supplied in the parameter variation file. SENSAN ensures that old model output files are deleted before the model is run so that, should this occur, out-of-date model outcome values are not read as current values. If, after the model has been run, a certain model output file is not found, SENSAN reports this condition to the screen, records the current set of parameter values to its output files, and moves on to the next parameter set. If a model run terminates prematurely for a particular parameter set and all model outcomes cannot be read from its output file(s), INSCHECK (run by SENSAN) will fail to produce an observation value file (which

SENSAN reads to ascertain model outcome values). Under these circumstances SENSAN reports to the screen that it cannot find an INSCHEK-generated observation value file, records the parameter values to its output file and moves on to the next parameter set.

Another reason why SENSAN may report that it cannot open a "temporary observation file" (i.e. an INSCHEK-generated file) is that it was unable to run INSCHEK and/or TEMPCHEK because the folders holding *inschek.exe* or *tempchek.exe* were not cited in the PATH environment variable. Alternatively, it may not have been able to run the model for the same reason.

If a parameter appears to be totally insensitive on SENSAN output files, make sure that it has been provided with the same name in the parameter variation file as that provided for this same parameter in any template file in which it appears. If parameter names are not identical between these two file types, some parameter values as supplied to SENSAN in the parameter variation file cannot be written to model input file(s). (Note, however, that such an error will be detected and recorded by SENSCHEK.)

When undertaking a SENSAN run for the first time, it is a good idea to set SCREENDISP to "verbose" so that TEMPCHEK and INSCHEK can report what they are doing to the screen. After any errors have been corrected, SCREENDISP can then be set to "noverbose" for routine SENSAN usage. Similarly, model output should not be directed to the null file until it is verified that SENSAN (through TEMPCHEK) is able to build correct input files for it. Failure in this regard will normally result in a model-generated error message. Conversely, if SENSAN or INSCHEK indicate a failure to read the model output file(s), a search should be made for a model-generated error message.

If running SENSAN in "verbose" mode for cases where there are multiple template files, the user may notice a message similar to the following scroll past on the screen.

```
Warning: parameter "ro1" from parameter value file t###.par not cited in
template file ves.tp2.
```

This is of no concern, for it is simply TEMPCHEK informing the user that it has been provided with a parameter value file (i.e. *t###.par* written by SENSAN) that contains the values of more parameters than are cited in any one template file.

17.7 An Example

Included in the *pestex* subfolder of the folder into which PEST is installed are the files required to run the soil clod shrinkage example discussed in chapter 18 of this manual. Also included in this subfolder are three files not discussed in chapter 18. These are *twofit.sns* a SENSAN control file, *out1.ins* an instruction file identical to *out.ins* discussed in chapter 18, and *parvar.dat* a parameter variation file.

An inspection of file *twofit.sns* reveals that this SENSAN control file assumes the same number of parameters and observations as does the PEST control file *twofit.pst*. As the parameter variation file *parvar.dat* reveals, parameter names are identical for the two cases. Also identical for the two cases are the template and instructions files; however the instruction file for the SENSAN example is named *out1.ins* instead of *out.ins* in order to avoid *out.obf* (used in the PEST example of chapter 18) being overwritten when SENSAN runs INSCHEK.

Five parameter sets are provided in *parvar.dat*, requiring that five model runs be undertaken. After the third model run has been completed, the model output file *out.dat* is copied to file

out.kp for safekeeping.

Before running SENSAN, make sure that the PEST folder is cited in the PATH environment variable (so that SENSAN can run TEMPCHEK and INSCHEK). Run SENSCHEK using the command

```
senschek twofit
```

After thus verifying that there are no errors or inconsistencies in the SENSAN input dataset, run SENSAN using the command

```
sensan twofit
```

After SENSAN has completed execution, inspect files *out1.txt*, *out2.txt* and *out3.txt*, the three SENSAN output files. You may also wish to verify that file *out.kp* exists, this being a record of *out.dat* generated on the third model run.

18. A Simple PEST Example

18.1 Parameter Estimation

18.1.1 Laboratory Data

This section takes you, step by step, through a simple example which demonstrates the application of PEST to a practical problem. Once PEST has been installed on your computer, the files cited in this chapter can be found in the *pestex* subfolder of the main PEST folder.

Table 18.1 shows the results of an experiment in which the specific volume of a soil clod (the reciprocal of its bulk density) is measured at a number of water contents as the clod is desiccated through oven heating. The data are plotted in figure 18.1; see also file *soilvol.dat*. We wish to fit two straight lines to this data. In soil physics parlance, the straight line segment of low slope fitted through the points of low water content is referred to as the "residual shrinkage" segment, whereas the segment covering the remainder of the data (with a slope near unity) is referred to as the "normal shrinkage" segment. (Actually, another segment of low slope is often present at high moisture contents, this being the "structural shrinkage" segment; this segment is not apparent in the data plotted in figure 18.1.)

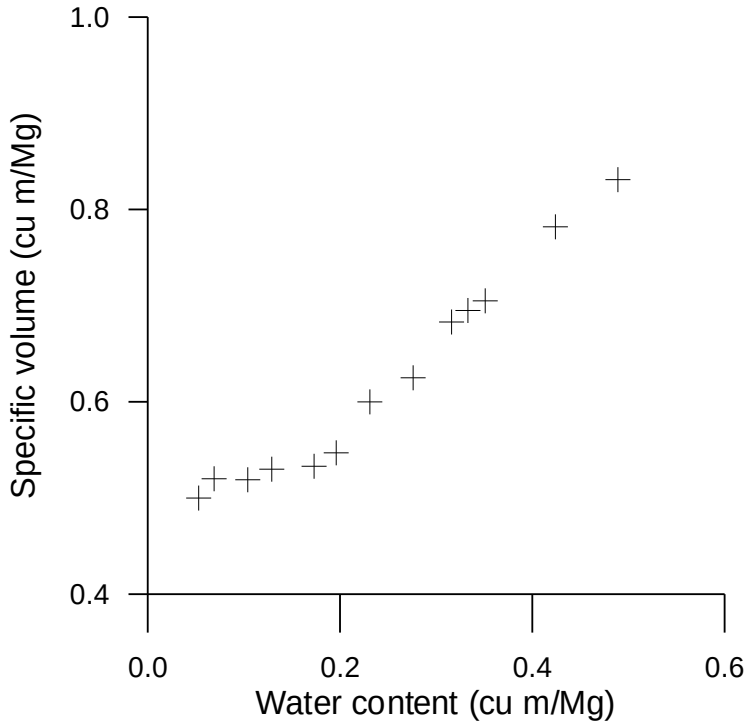


Figure 18.1 Soil clod shrinkage data.

water content (m^3/Mg)	specific volume (m^3/Mg)
0.052	0.501
0.068	0.521
0.103	0.520
0.128	0.531
0.172	0.534
0.195	0.548
0.230	0.601
0.275	0.626
0.315	0.684
0.332	0.696
0.350	0.706
0.423	0.783
0.488	0.832

Table 18.1 Soil clod shrinkage data.

18.1.2 The Model

Before we can use PEST there must be a model. We will list the model program in a moment; first we present the model algorithm.

Figure 18.2 shows two intersecting line segments. Let the slope of the first segment be s_1 and that of the second segment be s_2. Let the intercept of the first segment on the y-axis be y_1 and the x-coordinate of the point of intersection of the two line segments be x_c. The equation for the two-line system is

$$\begin{aligned} y &= s_1 x + y_1 && x \le x_c \\ y &= s_2 x + (s_1 - s_2) x_c + y_1 && x > x_c \end{aligned} \qquad (18.1.1)$$

where x is the water content and y represents the soil clod specific volume.

A simple FORTRAN program can be written based on this concept; a listing is provided in figure 18.3 (see also file *twoline.for* in the *pestex* subfolder). Program TWOLINE begins by reading an input file named *in.dat* which supplies it with values for s_1, s_2, y_1 and x_c, as well as water contents (i.e. x values in equation 18.1.1) at which soil clod specific volumes are required. TWOLINE writes a single output file (named *out.dat*) listing both water contents and the specific volumes calculated for these water contents. Figure 18.4 shows a typical TWOLINE input file, while figure 18.5 shows a corresponding TWOLINE output file. An executable version of TWOLINE (namely *twoline.exe*) is provided in the *pestex* subfolder of the PEST installation folder.

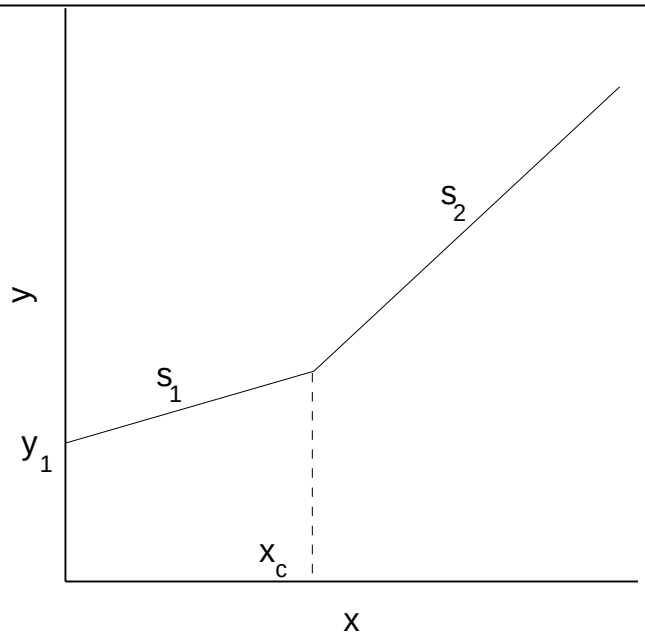


Figure 18.2 Parameters of the two line model.

```
      program twoline

      integer*4 i,nx
      real*4 s1,s2,y1,xc
      real*4 x(50),y(50)

      open(unit=20,file='in.dat')

c     read the line parameters

      read(20,*) s1,s2
      read(20,*) y1
      read(20,*) xc

c     read the abscissae at which there are measurement values

      read(20,*) nx
      do 100 i=1,nx
      read(20,*) x(i)
100   continue
      close(unit=20)

c     evaluate y for each x

      do 200 i=1,nx
      if(x(i).le.xc) then
        y(i)=s1*x(i)+y1
      else
        y(i)=s2*x(i)+(s1 s2)*xc+y1
      end if
200   continue

c     write the y values to the output file

      open(unit=20,file='out.dat')
      do 300 i=1,nx
      write(20,*) x(i),y(i)
300   continue
      close(unit=20)

      end
```

Figure 18.3 A listing of program TWOLINE.

We would like TWOLINE to calculate specific volumes at water contents corresponding to our experimental dataset as set out in table 18.1. The input file of figure 18.4 ensures that this will, indeed, occur. Hence TWOLINE is now our system model. We would like PEST to adjust the parameters of this model such that the discrepancies between laboratory and model-generated specific volumes are as small as possible. The parameters in this case are the four line parameters, namely s_1, s_2, y_1 and x_c. Now that our model is complete, our next task is to prepare the TWOLINE-PEST interface.

```
0.3  0.8
0.4
0.3
13
0.052
0.068
0.103
```

```
0.128
0.172
0.195
0.230
0.275
0.315
0.332
0.350
0.423
0.488
```

Figure 18.4 A TWOLINE input file *in.dat*.

```
 0.520000E 01   0.415600
 0.680000E 01   0.420400
 0.103000       0.430900
 0.128000       0.438400
 0.172000       0.451600
 0.195000       0.458500
 0.230000       0.469000
 0.275000       0.482500
 0.315000       0.502000
 0.332000       0.515600
 0.350000       0.530000
 0.423000       0.588400
 0.488000       0.640400
```

Figure 18.5 A TWOLINE output file *out.dat*.

18.1.3 Preparing the Template File

First a template file must be prepared. This is easily accomplished by copying the file *in.dat* listed in figure 18.4 to the file *in.tpl* and modifying this latter file in order to turn it into a PEST template file. Figure 18.6 shows the resulting template file; the value of each of the line parameters has been replaced by an appropriately named parameter space, and the "ptf" header line has been added to the top of the file. Because TWOLINE reads all parameters using free field format, the width of each parameter space is not critical; however where two parameters are found on the same line, they must be separated by a space. A parameter space width of 13 characters is employed in file *in.tpl* in order to use the maximum precision available for representing single precision numbers. As discussed in section 2.2.5, while PEST does not insist that parameters be written with maximum precision to model input files, it is a good idea nevertheless.

```
ptf #
# s1         # # s2         #
# y1         #
# xc         #
13
0.052
0.068
0.103
0.128
0.172
0.195
0.230
0.275
0.315
```

```
0.332
0.350
0.423
0.488
```

Figure 18.6 The template file *in.tpl*.

Now that *in.tpl* has been prepared, it should be checked using program TEMPCHEK; run TEMPCHEK using the command

```
tempchek in.tpl
```

Figure 18.7 shows file *in.pmt*, written by TEMPCHEK, in which all parameters cited in file *in.tpl* are listed. By copying file *in.pmt* to *in.par* and adding parameter values, SCALEs and OFFSETs to the listed parameter names, as well as values for the character variables PRECIS and DPOINT, we can create a PEST parameter value file. Figure 18.8 shows such a file; because this file will shortly be used with program PESTGEN (see part II of this manual) to generate a PEST control file, the values supplied for each of the parameters are the initial parameter values to be used in the parameter estimation process.

```
s1
s2
y1
xc
```

Figure 18.7 File *in.pmt*.

```
single point
 s1  0.3  1.0 0.0
 s2  0.8  1.0 0.0
 y1  0.4  1.0 0.0
 xc  0.3  1.0 0.0
```

Figure 18.8 File *in.par*.

At this stage TEMPCHEK should be run again using the command

```
tempchek in.tpl in.dat in.par
```

When run using this command, TEMPCHEK generates file *in.dat*, the TWOLINE input file, using the parameter values provided in file *in.par*; you should then run TWOLINE, making sure that it reads this file correctly.

18.1.4 Preparing the Instruction File

Next the instruction file should be prepared. This can be easily accomplished by writing the instructions shown in figure 18.9 to file *out.ins* using a text editor. Using this instruction set, all model-generated observations are read as semi-fixed observations; while they could have been read as fixed observations, we may have been unsure of just how wide a number can ever get in the second column of file *out.dat* (for example if a number becomes negative, very large or very small).

```
pif #
l1 (o1)19:26
l1 (o2)19:26
l1 (o3)19:26
l1 (o4)19:26
l1 (o5)19:26
l1 (o6)19:26
```

```
l1 (o7)19:26
l1 (o8)19:26
l1 (o9)19:26
l1 (o10)19:26
l1 (o11)19:26
l1 (o12)19:26
l1 (o13)19:26
```

Figure 18.9 The instruction file *out.ins*.

Program INSCHEK should now be used to check that file *out.ins* contains a legal instruction set. Run INSCHEK using the command

```
inschek out.ins
```

If no errors are encountered you should then run INSCHEK again, this time directing it to read a TWOLINE output file using the instruction set; use the command

```
inschek out.ins out.dat
```

INSCHEK will produce a file named *out.obf* listing the values it reads from file *out.dat* for the observations cited in file *out.ins*; see figure 18.10.

```
 o1       0.415600
 o2       0.420400
 o3       0.430900
 o4       0.438400
 o5       0.451600
 o6       0.458500
 o7       0.469000
 o8       0.482500
 o9       0.502000
 o10      0.515600
 o11      0.530000
 o12      0.588400
 o13      0.640400
```

Figure 18.10 file *out.obf*.

18.1.5 Preparing the PEST Control File

The PEST-TWOLINE interface is now complete as PEST can now generate a TWOLINE input file and read a TWOLINE output file. The next step is to generate a PEST control file through which PEST is provided with an appropriate set of control variables and in which the laboratory measurements of specific volume are provided. First copy file *out.obf* to file *measure.obf*. Then replace the value of each model-generated observation with the corresponding value from table 18.1, i.e. with the appropriate laboratory measurement; see figure 18.11. Then run PESTGEN using the command

```
pestgen twofit in.par measure.obf
```

```
   o1   0.501
   o2   0.521
   o3   0.520
   o4   0.531
   o5   0.534
   o6   0.548
   o7   0.601
   o8   0.626
   o9   0.684
   o10  0.696
```

```
   o11  0.706
   o12  0.783
   o13  0.832
```

Figure 18.11 File *measure.obf*.

PESTGEN generates a PEST control file named *twofit.pst*; see figure 18.12. File *twofit.pst* should now be edited as some of the default values used by PESTGEN in writing this file are not appropriate to our problem. In particular, our model is run using the command "twoline", not "model"; the filenames listed in the "model input/output" section of *twofit.pst* need to be altered as well. Once you have made these changes (figure 18.13 lists that part of *twofit.pst* to which the alterations have been made), preparation for the PEST run is complete. It would be a very good idea to make some other adjustments to *twofit.pst* as well, such as providing more appropriate upper and lower bounds for each of the parameters. However, at the risk of leading you into bad habits, this will not be done.

```
pcf
* control data
restart estimation
    4    13     4     0     1
    1     1 single point   1  0  0
  5.0   2.0   0.3  0.03    10
  3.0   3.0 0.001
  0.1
   30  0.01     3     3  0.01     3
    1     1     1
* parameter groups
s1   relative 0.01  0.0  switch  2.0 parabolic
s2   relative 0.01  0.0  switch  2.0 parabolic
y1   relative 0.01  0.0  switch  2.0 parabolic
xc   relative 0.01  0.0  switch  2.0 parabolic
* parameter data
s1   none relative   0.300000  1.00000E+10   1.00000E+10  s1   1.0000  0.000  1
s2   none relative   0.800000  1.00000E+10   1.00000E+10  s2   1.0000  0.000  1
y1   none relative   0.400000  1.00000E+10   1.00000E+10  y1   1.0000  0.000  1
xc   none relative   0.300000  1.00000E+10   1.00000E+10  xc   1.0000  0.000  1
* observation groups
obsgroup
* observation data
o1    0.501000      1.0  obsgroup
o2    0.521000      1.0  obsgroup
o3    0.520000      1.0  obsgroup
o4    0.531000      1.0  obsgroup
o5    0.534000      1.0  obsgroup
o6    0.548000      1.0  obsgroup
o7    0.601000      1.0  obsgroup
o8    0.626000      1.0  obsgroup
o9    0.684000      1.0  obsgroup
o10   0.696000      1.0  obsgroup
o11   0.706000      1.0  obsgroup
o12   0.783000      1.0  obsgroup
o13   0.832000      1.0  obsgroup
* model command line
model
* model input/output
model.tpl  model.inp
model.ins  model.out
* prior information
```

Figure 18.12 The PESTGEN-generated control file *twofit.pst*.

```
model command line
```

```
twoline
* model input/output
in.tpl  in.dat
out.ins  out.dat
```

Figure 18.13 Altered sections of *twofit.pst*.

As a final check that the entire PEST input dataset is complete, correct and consistent, you should run program PESTCHEK using the command

```
pestchek twofit
```

If all is correct, you can now run PEST using the command

```
pest twofit
```

A run record file *twofit.rec* is written by PEST in the *pestex* subfolder; so too is file *twofit.par* containing the estimated parameter set. Figure 18.14 shows the lines of best fit superimposed on the laboratory data.

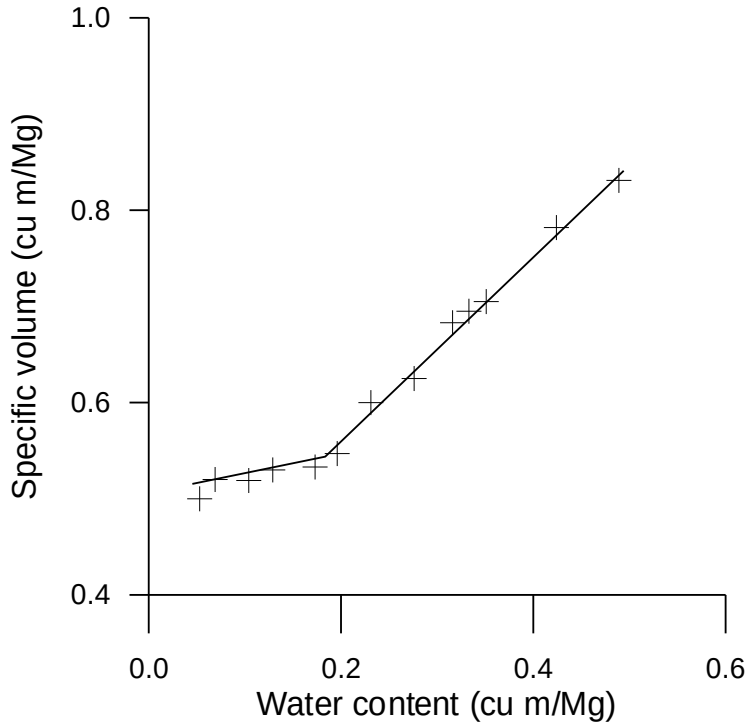


Figure 18.14 Soil clod shrinkage data with lines of best fit superimposed.

18.2 Predictive Analysis

18.2.1 Obtaining the Model Prediction of Maximum Likelihood

Files for this example can be found in the *papestex* subfolder of the PEST folder after installation. This example builds on the soil clod shrinkage example discussed in the previous section.

Based on the dataset supplied with the previous example, PEST lowers the objective function to a value of 6.71E-4 when estimating values for the model parameters. Best fit parameter values are listed in table 18.2.

Parameter	PEST-estimated value
s1	0.238
s2	0.963
y1	0.497
xc	0.174

Table 18.2 Optimised parameter values for soil clod shrinkage example.

After PEST is run in "estimation" mode, best-fit model parameters can be found in file *twofit.par*. Insert these into the model input file *in.dat* by running TEMPCHEK as follows.

```
tempchek in.tpl in.dat twofit.par
```

If the value for specific volume at a water content of 0.4 is of particular interest to us, this can now be easily calculated using our calibrated "model". An appropriate TWOLINE input file *in2.dat* is provided; this is easily prepared from the new *in.dat* file created using TEMPCHEK by alteration with a text editor in conformity with the expectations of program TWOLINE. As the first two lines of this file contain parameter values to be used by TWOLINE, it was necessary to run TEMPCHEK first to ensure that the parameter values contained in this file are the optimal parameter values.

As TWOLINE expects a file named *in.dat*, *in2.dat* must be copied to *in.dat* before TWOLINE is run. But before doing this, copy the existing *in.dat* to *in1.dat* for safekeeping.

After running program TWOLINE (by typing "twoline" at the screen prompt) open file *out.dat* to obtain the model-predicted specific volume. It should be 0.756. This is thus our best estimate of the soil clod specific volume at a water content of 0.4.

18.2.2 The Composite Model

Before undertaking predictive analysis, we must construct a "composite model" comprised of the model run under calibration conditions followed by the model run under prediction conditions. This model must be encompassed in a batch file; an appropriate file named *model.bat* is provided. Figure 18.15 shows a printout of *model.bat.*

```
@echo off

rem Intermediate files are deleted

del in.dat
del out.dat

rem First the model is run under calibration conditions.

copy in1.dat in.dat > nul
twoline > nul
copy out.dat out1.dat > nul

rem Next the model is run under predictive conditions.

copy in2.dat in.dat > nul
twoline > nul
copy out.dat out2.dat > nul
```

Figure 18.15 A batch file encompassing the composite model.

The batch file is divided into three parts. In the second part the model is run under calibration conditions. First the "calibration input file" *in1.dat* is copied to the expected TWOLINE input file *in.dat*. TWOLINE is then run and its output file *out.dat* is copied to another file *out1.dat* for safekeeping.

The process is then repeated in the third part of file *model.bat* for the predictive model run. In this case *in2.dat* is the model input file and *out2.dat* is the model output file.

The first part of the batch file *model.bat* illustrates a procedure that is recommended in the construction of all composite models. In this section of the batch file all intermediate files used or produced during execution of the composite model are deleted. Recall that PEST deletes all model output files (that it knows about) before it runs the model. In this way it is ensured that if, for some reason, the model does not run, then old model output files are not mistaken for new ones. Thus if the model fails to run, an error condition will be encountered and PEST will cease execution with an appropriate error message. However in a composite model there are likely to be intermediate files which are generated by one model to be read by another. If any part of a composite model fails to execute, the ensuing part of the composite model must be prevented from executing on the basis of intermediate files generated during previous model runs. This can be ensured by deleting all such intermediate files.

The first line of the batch file *model.bat* prevents the operating system from echoing batch file commands to the screen. This relieves screen clutter when the model is run under the control of PEST in the same window as PEST. To assist in this process screen output from all commands is directed to the null file (i.e. file *nul* in the present case) instead of to the screen.

To satisfy yourself that the composite model runs correctly, type

```
model
```

at the screen prompt. Inspect the model output files *out1.dat* and *out2.dat*. (You may wish to delete the "@echo off" line and remove "> nul" from each command line before you run the model in order to see the appropriate model commands scroll past on the screen as they are executed.)

18.2.3 The PEST Control File

We would now like to obtain the maximum possible value for the soil specific volume at a water content of 0.4 compatible with the model being calibrated against our laboratory dataset. Let us do this by assuming that the model can still be considered to be calibrated if the objective function under calibration conditions is as high as 5.0E-3; this is thus the value assigned to the predictive analysis control variable PD0.

The PEST control file used in the parameter estimation process was named *twofit.pst*. This file should be copied to file *twofit1.pst* and the following alterations made (actually this has already been done for you).

1. Replace the word "estimation" with the word "prediction" on the third line of this file.
2. When undertaking a predictive analysis run there is an extra observation, this being the model prediction. Hence the number of observations (i.e. NOBS) must be increased from 13 to 14 on the fourth line of file *twofit1.pst*.
3. There will now be two observation groups, so alter the fifth entry on line four of file

twofit1.pst (i.e. NOBSGP) to 2.

4. There are now two model input files and two model output files, so alter the first two entries on the fifth line of file *twofit1.pst* (i.e. NTPLFLE and NINSFLE) to 2 and 2 respectively.
5. In the "observation groups" section of the PEST control file add the observation group "predict".
6. In the "observation data" section of the PEST control file add an extra observation named "o14". Assign this to the observation group "predict". Provide whatever observation value and weight that you like, as these will be ignored by PEST. It is probably best to make both of these 0.0, just in case you wish to run PEST later using the same file in "estimation" mode; by assigning the weight as zero, the "prediction observation" will contribute nothing to the objective function if that occurs.
7. Alter the model command line to "*model.bat*" in the "model command line" section of the new PEST control file.
8. The two model input files are named *in1.dat* and *in2.dat*; the first is used for the calibration component of the composite model, the second is used for the predictive component of the composite model. A PEST template file already exists for the first model input file (i.e. *in.tpl*). We will introduce a new template file for the second model input file shortly; it will be called *in2.tpl*. So alter the model input filename to *in1.dat* on the first line of the "model command line" section of file *twofit1.pst*; then add another line beneath this comprised of the entries "*in2.tpl*" and "*in2.dat*".
9. The model output files are named *out1.dat* and *out2.dat*; the first is produced by the calibration component of the composite model while the second is produced by the predictive component of the composite model. A PEST instruction file already exists for the first model output file (i.e. *out.ins*). We will introduce a new instruction file for the second one shortly; it will be called *out2.ins*. So alter the model output filename to "*out1.dat*" on the third line of the "model command line" section of file *twofit1.pst*; then add another line underneath this comprised of the entries "*out2.ins*" and "*out2.dat*".
10. Add a "predictive analysis" section to file *twofit1.pst*. Because we wish to maximise the prediction, NPREDMAXMIN is assigned the value of 1. As mentioned above, PD0 is 5.0E-3. Set PD1 to 5.2E-3 and set PD2 to be twice as high as PD0, i.e. 1.0E-2. Variables governing operation of the Marquardt lambda, the switching from two point to three point derivatives calculation, and the termination of execution will be set in relative rather than absolute terms, so set ABSPREDLAM, ABSPREDSWH and ABSPREDSTP to 0.0. RELPREDLAM, RELPREDSWH and RELPREDSTP should be set to their recommended values of .005, .05 and .005 respectively. NPREDNORED and NPREDSTP should be set at their recommended values of 4 and 4. Contrary to the advice provided in chapter 8, we will not conduct a line search for each Marquardt lambda, so set the control variables INITSCHFAC, MULSHFAC and NSEARCH to 1.0, 2.0 and 1 respectively.

18.2.4 Template and Instruction Files

Inspect files *in2.tpl* and *out2.ins* provided with the example files. These are, respectively, a template file for *in2.dat* and an instruction file to read the single prediction observation from file *out2.dat*.

Notice how parameter spaces for each of the four parameters involved in the predictive analysis process appear in both of files *in.tpl* and *in2.tpl*. This is because these parameter

values are used by the model under both calibration and prediction conditions. Prior to running the composite model they must be written to both sets of input files (together with other data specific to each component of the composite model).

18.2.5 Running PEST

Before running PEST, run PESTCHEK to check that the entire input dataset is consistent and correct. At the screen prompt type

```
pestchek twofit1
```

Then run PEST using the command

```
pest twofit1
```

There are two things to watch as PEST executes. The first is the value of the objective function and the second is the value of the prediction. Both of these are written to the screen on every occasion that PEST calculates a parameter upgrade vector (these are easily seen when running PEST if screen output from the composite model is disabled as discussed earlier). The objective function (i.e. phi) hovers around 5.0E-3 as it should (though values on either side of this are recorded). The value of the prediction slowly rises from iteration to iteration. Note that information written to the screen during the course of PEST's execution is also recorded in the PEST run record file (in this case *twofit1.rec*).

When PEST ceases execution, open file *twofit1.rec* and go to the bottom of the file. Near the bottom of the file it is written that PEST achieved a maximum prediction value of 0.786 for a corresponding objective function value of 5.16E-3. This is a little above our target value of 5.0E-3, but is accepted due to the action of PD1. However due to the rather subjective way in which an objective function value was selected at which the model is said to be "calibrated" this matters little.

While inspecting the run record file, notice how observation "o14" is not listed with other observations in the section of this file which tabulates observed values, corresponding model-generated values and residuals. This is because observation "o14" is in fact the prediction, PEST recognising it as such because it is the only observation assigned to the observation group "predict".

Figure 18.16 shows a plot of the line segments calculated on the basis of the parameters derived by PEST during the above predictive analysis process. The fit is not too bad, though obviously not as good as that obtained on the basis of best fit parameters.

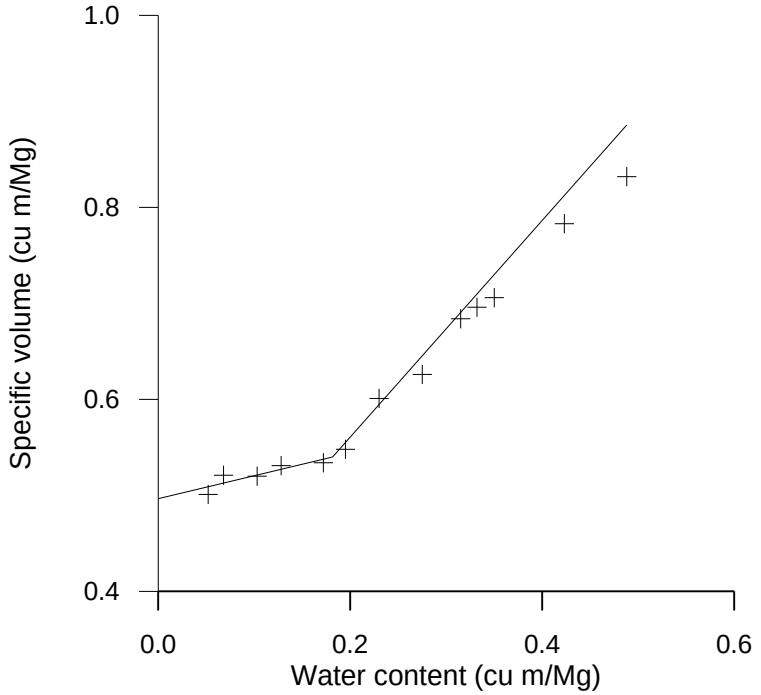


Figure 18.16 Soil clod shrinkage data with line segments superimposed.

In the present instance, the "worst case" model prediction of 0.786 is not too different from the "most likely" model prediction of 0.756. This is comforting to know. It is a frightening fact that in many instances of environmental modelling the worst case prediction can be hugely different from that calculated using parameters corresponding to the objective function minimum. It is under these circumstances that predictive analysis becomes an absolute necessity.

19. References

Banta, E.R., Poeter, E.P, Doherty, J.E., and Hill, M.C., 2006. JUPITER: Joint Universal Parameter IdenTification and Evaluation of Reliability - An Application Programming Interface (API) for Model Analysis: U.S. Geological Survey Techniques and Methods 6–E1, 268 p.

Bayer, P. and M. Finkel, 2007. Optimisation of concentration control by evolution strategies: formulation, application and assessment of remedial solutions. *Water Resourc. Res.* 43: doi: 10.1029/2005WR004753.

Burrows, W. and Doherty, J., 2015. Efficient calibration/uncertainty analysis using paired complex/simple models. *Groundwater*, 53(4), 531-541.

Christensen, S. and Cooley, R.L. 1999. Evaluation of prediction intervals for expressing uncertainties in groundwater flow model predictions. *Water Resour. Res.*, 35 (9), 2627-2639.

Cooley, R.L., 1983. Some new procedures for numerical solution of variably saturated flow problems. *Water Resour. Res.*, 19 (5), pp1271-1285.

Cooley, R.L. and Vecchia, A.V., 1987. Calculation of nonlinear confidence and prediction intervals for ground-water flow models. *Water Resources Bulletin*. 23 (4), 581-599.

Doherty, J., 2015. Calibration and uncertainty analysis for complex environmental models. *Watermark Numerical Computing*, Brisbane, Australia. 227pp. ISBN: 978-0-9943786-0-6 Downloadable from www.pesthomepage.org.

Doherty, J. and Simmons, C.T., 2013. Groundwater modelling in decision support: reflections on a unified conceptual framework. *Hydrogeology Journal* 21, 1531–1537

Doherty, J. and Vogwill, R., 2015. Models, Decision-Making and Science. In *Solving the Groundwater Challenges of the 21st Century*. Vogwill, R. editor. CRC Press. In Press.

Duan, Q., 1991. A global optimization strategy for efficient and effective calibration of hydrologic models. PhD thesis, Department of Hydrology and Water Resources, University of Arizona, Tuscon.

Duan, Q., Soorooshian, S. and Gupta, V., 1992. Effective and efficient global optimization for conceptual rainfall-runoff models. *Water Resourc. Res.*, 28 (4), 1015-1031.

Duan, Q., Gupta, V.K. and Sorooshian, S., 1993. A Shuffled Complex Evolution approach for effective and efficient global minimization. *Journal of Optimization Theory and its Applications*, 76 (3), 501-521.

Duan, Q., Sorooshian, S. and Gupta, V.K., 1994. Optimal use of the SCE-UA global optimization method for calibrating watershed models. *J. Hydrol.*, 158, 265-284.

Hansen, N. and A. Ostermeier, 2001. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9, 159-195

Hansen, N., S.D. Muller, and P. Koumoutsakos, 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol. Comput.*, 9, 159-195.

Hill, M. C., 1992. A Computer Program (MODFLOWP) for Estimating Parameters of a Transient, Three-Dimensional, Ground-Water Flow Model using Nonlinear Regression. U.

S. Geological Survey Open-File Report 91-484.

Kavetski, D., Kuczera, G., and Franks, S.W., 2006. Calibration of conceptual hydrological models revisited: 1. overcoming numerical artifacts. *J. Hydrol., 320*, 173-186.

Nearing, J., 2001. Mathematical Tools for Physics. Department of Physics, University of Miami. Downloadable from http://www.physics.miami.edu/~nearing/mathmethods/mathematical_methods-three.pdf

Paige, C.C. and Saunders, M.A., 1982a. LSQR: an algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, vol8: 43-71.

Paige, C.C. and Saunders, M.A. 1982b. LSQR: an algorithm for spare linear equations and sparse least squares. *ACM Trans. Math. Softw*., Vol 8: 192-209.

Vecchia, A.V. and Cooley, R.L., 1987. Simultaneous confidence and prediction intervals for nonlinear regression models with application to a ground water flow model. *Water Resour. Res.*, 23, 7, 1237-1250.

White, J.T., Doherty, J.E. and Hughes, J.D., 2014. Quantifying the predictive consequences of model error with linear subspace analysis. *Water Resour. Res*, 50 (2): 1152-1173. DOI: 10.1002/2013WR014767

Appendix A. PEST Control File Specifications

This appendix provides specifications for a PEST control file.

Variables are recognized by their position in the file. They must be placed on the correct line of this file and separated from their neigbours by at least one space.

PEST, BEOPEST and many of the PEST-support utility programs which are documented in part II of this manual tolerate the presence of the following items in a PEST control file:

- blank lines;
- comments;
- lines that begin with the character string "++".

Lines that begin with "++" are used for the insertion of variables which control the operation of the PEST++ suite of programs.

Comments can be placed on their own line. Alternatively, they can be placed at the end of a line which provides PEST control data. In either case, a comment follows a "#" character. Note, however, that this character is not construed as denoting the presence of an ensuing comment under any of the following circumstances:

- it is not prededed by a space, tab or the beginning of a line;
- it is part of a string that is enclosed in quotes.

These exceptions preclude mis-construing the presence of the "#" character in a filename as the start of a comment.

Some of the older utilities that are documented in part II of this manual will not tolerate the presence of blank lines, "++" lines or comments. All of these items can be removed from a PEST control file using the PSTCLEAN utility.

Figure A1.1 Names and locations of variables in the PEST control file. Optional variables are enclosed in square brackets.

```
pcf
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM][DERZEROLIM]
NTPLFLE NINSFLE PRECIS DPOINT [NUMCOM JACFILE MESSFILE] [OBSREREF]
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE] [LAMFORGIVE] [DERFORGIVE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND] [ABSPARMAX]
PHIREDSWH [NOPTSWITCH] [SPLITSWH] [DOAUI] [DOSENREUSE] [BOUNDSCALE]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN] [PHIABANDON]
ICOV ICOR IEIG [IRES] [JCOSAVE] [VERBOSEREC] [JCOSAVEITN] [REISAVEITN] [PARSAVEITN] [PARSAVERUN]
* sensitivity reuse
SENRELTHRESH  SENMAXREUSE
SENALLCALCINT  SENPREDWEIGHT  SENPIEXCLUDE
* singular value decomposition
SVDMODE
MAXSING EIGTHRESH
EIGWRITE
* lsqr
LSQRMODE
LSQR_ATOL LSQR_BTOL LSQR_CONLIM LSQR_ITNLIM
LSQRWRITE
* automatic user intervention
MAXAUI AUISTARTOPT NOAUIPHIRAT AUIRESTITN
AUISENSRAT AUIHOLDMAXCHG AUINUMFREE
AUIPHIRATSUF AUIPHIRATACCEPT NAUINOACCEPT
* svd assist
BASEPESTFILE
BASEJACFILE
SVDA_MULBPA SVDA_SCALADJ SVDA_EXTSUPER SVDA_SUPDERCALC SVDA_PAR_EXCL
* parameter groups
PARGPNME  INCTYP  DERINC  DERINCLB  FORCEN  DERINCMUL  DERMTHD  [SPLITTHRESH  SPLITRELDIFF
SPLITACTION]
(one such line for each of NPARGP parameter groups)
* parameter data
PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM
(one such line for each of NPAR parameters)
PARNME PARTIED
(one such line for each tied parameter)
* observation groups
OBGNME [GTARG] [COVFLE]
(one such line for each of NOBSGP observation groups)
* observation data
OBSNME OBSVAL WEIGHT OBGNME
(one such line for each of NOBS observations)
* derivatives command line
DERCOMLINE
EXTDERFLE
* model command line
COMLINE
(one such line for each of NUMCOM command lines)
* model input/output
TEMPFLE INFLE
(one such line for each of NTPLFLE template files)
INSFLE OUTFLE
(one such line for each of NINSLFE instruction files)
* prior information
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of NPRIOR articles of prior information)
* predictive analysis
NPREDMAXMIN [PREDNOISE]
PD0 PD1 PD2
ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH
ABSPREDSWH RELPREDSWH
NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP
* regularisation
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE]
WFINIT  WFMIN  WFMAX  [LINREG] [REGCONTINUE]
WFFAC  WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]
* pareto
PARETO_OBSGROUP
```

```
PARETO_WTFAC_START PARETO_WTFAC_FIN NUM_WTFAC_INC
NUM_ITER_START NUM_ITER_GEN NUM_ITER_FIN
ALT_TERM
OBS_TERM ABOVE_OR_BELOW OBS_THRESH NUM_ITER_THRESH (only if ALT_TERM is non-zero)
NOBS_REPORT
OBS_REPORT_1 OBS_REPORT_2 OBS_REPORT_3.. (NOBS_REPORT items)
```

Table A1.1 Variables in the "control data" section of the PEST control file.

Variable	Type	Values	Description
RSTFLE	text	"restart" or "norestart"	instructs PEST whether to write restart data
PESTMODE	text	"estimation", "prediction", "regularisation", "pareto"	PEST's mode of operation
NPAR	integer	greater than zero	number of parameters
NOBS	integer	greater than zero	number of observations
NPARGP	integer	greater than zero	number of parameter groups
NPRIOR	integer	any integer value	absolute value is number of prior information equations; negative value indicates supply of prior information in indexed format
NOBSGP	integer	greater than zero	number of observation groups
MAXCOMPDIM	integer	zero or greater	number of elements in compressed Jacobian matrix
NTPLFLE	integer	greater than zero	number of template files
NINSFLE	integer	greater than zero	number of instruction files
PRECIS	text	"single" or "double"	format for writing parameter values to model input files
DPOINT	text	"point" or "nopoint"	omit decimal point in parameter values if possible
NUMCOM	integer	greater than zero	number of command lines used to run model
JACFILE	integer	0, 1 or -1	indicates whether model provides external derivatives file
MESSFILE	integer	zero or one	indicates whether PEST writes PEST-to-model message file
OBSREREF	text	"obsreref", "obsreref_N" or "noobsreref"	activates or de-activates observation re-referencing (with an optional pause after re-referencing runs)
RLAMBDA1	real	zero or greater	initial Marquardt lambda
RLAMFAC	real	positive or negative, but not zero	dictates Marquardt lambda adjustment process
PHIRATSUF	real	between zero and one	fractional objective function sufficient for end of current iteration
PHIREDLAM	real	between zero and one	termination criterion for Marquardt lambda search
NUMLAM	integer	one or greater; possibly negative with Parallel or BEOPEST	maximum number of Marquardt lambdas to test

JACUPDATE	integer	zero or greater	activation of Broyden's Jacobian update procedure
LAMFORGIVE	text	"lamforgive" or "nolamforgive"	treat model run failure during lambda search as high objective function
DERFORGIVE	text	"derforgive" or "noderforgive"	accomodates model failure during Jacobian runs by setting pertinent sensitivities to zero
RELPARMAX	real	greater than zero	parameter relative change limit
FACPARMAX	real	greater than one	parameter factor change limit
FACORIG	real	between zero and one	minimum fraction of original parameter value in evaluating relative change
ABSPARMAX(*N*)	real	greater than zero	parameter absolute change limit – *N*'th instance
IBOUNDSTICK	integer	zero or greater	instructs PEST not to compute derivatives for parameter at its bounds
UPVECBEND	integer	zero or one	instructs PEST to bend parameter upgrade vector if parameter hits bounds
PHIREDSWH	real	between zero and one	sets objective function change for introduction of central derivatives
NOPTSWITCH	integer	one or greater	iteration before which PEST will not switch to central derivatives computation
SPLITSWH	real	zero or greater	the factor by which the objective function rises to invoke split slope derivatives analysis until end of run
DOAUI	text	"aui", "auid", or "noaui"	instructs PEST to implement automatic user intervention
DOSENREUSE	text	"senreuse" or "nosenreuse"	instructs PEST to reuse parameter sensitivities
BOUNDSCALE	text	"boundscale" or "noboundscale"	parameters are scaled by the inter-bounds interval if using singular value decomposition, LSQR or SVDA
NOPTMAX	integer	-2, -1, 0, or any number greater than zero	number of optimisation iterations
PHIREDSTP	real	greater than zero	relative objective function reduction triggering termination
NPHISTP	integer	greater than zero	number of successive iterations over which PHIREDSTP applies
NPHINORED	integer	greater than zero	number of iterations since last drop in objective function to trigger termination
RELPARSTP	real	greater than zero	maximum relative parameter change triggering termination

NRELPAR	integer	greater than zero	number of successive iterations over which RELPARSTP applies
PHISTOPTHRESH	real	zero or greater	objective function threshold triggering termination
LASTRUN	integer	zero or one	instructs PEST to undertake (or not) final model run with best parameters
PHIABANDON	real or text	a positive number or name of a file	objective function value at which to abandon optimisation process or filename containing abandonment schedule
ICOV	integer	zero or one	record covariance matrix in matrix file
ICOR	integer	zero or one	record correlation coefficient matrix in matrix file
IEIG	integer	zero or one	record eigenvectors in matrix file
IRES	integer	zero or one	record resolution data
JCOSAVE	text	“jcosave” or “nojcosave”	save best Jacobian file as a JCO file - overwriting previously-saved files of the same name as the inversion process progresses
VERBOSEREC	text	“verboserec” or “noverboserec”	if set to “noverboserec”, parameter and observation data lists are omitted from the run record file
JCOSAVEITN	text	“jcosaveitn” or “nojcosaveitn”	write current jacobian matrix to iteration-specific JCO file at the end of every optimisation iteration
REISAVEITN	text	“reisaveitn” or “noreisaveitn”	store best-fit residuals to iteration-specific residuals file at end of every optimisation iteration
PARSAVEITN	text	“parsaveitn” or “noparsaveitn”	store iteration specific parameter value files
PARSAVERUN	text	“parsaverun” or “noparsaverun”	store run specific parameter value files

Table A1.2 Variables in the optional "sensitivity reuse" section of the PEST control file.

Variable	Type	Values	Description
SENRELTHRESH	real	zero to one	relative parameter sensitivity below which sensitivity reuse is activated for a parameter
SENMAXREUSE	integer	integer other than zero	maximum number of reused sensitivities per iteration
SENALLCALCINT	integer	greater than one	iteration interval at which all sensitivities re-calculated
SENPREDWEIGHT	real	any number	weight to assign to prediction in computation of composite parameter sensitivities to determine sensitivity reuse
SENPIEXCLUDE	text	"yes" or "no"	include or exclude prior information when computing composite parameter sensitivities to determine sensitivity reuse

Table A1.3 Variables in the optional "automatic user intervention" section of the PEST control file.

Variable	Type	Values	Description
MAXAUI	integer	zero or greater	maximum number of AUI iterations per optimisation iteration
AUISTARTOPT	integer	one or greater	optimisation iteration at which to commence AUI
NOAUIPHIRAT	real	between zero and one	relative objective function reduction threshold triggering AUI
AUIRESTITN	integer	zero or greater, but not one	AUI rest interval expressed in optimisation iterations
AUISENSRAT	real	greater than one	composite parameter sensitivity ratio triggering AUI
AUIHOLDMAXCHG	integer	zero or one	instructs PEST to target parameters which change most when deciding which parameters to hold
AUINUMFREE	integer	greater than zero	cease AUI when only AUINUMFREE parameters are unheld
AUIPHIRATSUF	real	between zero and one	relative objective function improvement for termination of AUI
AUIPHIRATACCEPT	real	between zero and one	relative objective function reduction threshold for acceptance of AUI-calculated parameters
NAUINOACCEPT	integer	greater than zero	number of iterations since acceptance of parameter change for termination of AUI

Table A1.4 Variables in the optional "singular value decomposition" section of the PEST control file.

Variable	Type	Values	Description
SVDMODE	integer	zero or one	activates truncated singular value decomposition for solution of inverse problem
MAXSING	integer	greater than zero	number of singular values at which truncation occurs
EIGTHRESH	real	zero or greater, but less than one	eigenvalue ratio threshold for truncation
EIGWRITE	integer	zero or one	determines content of SVD output file

Table A1.5 Variables in the optional "LSQR" section of the PEST control file.

Variable	Type	Values	Description
LSQRMODE	integer	zero or one	activates LSQR solution of inverse problem
LSQR_ATOL	real	zero or greater	LSQR algorithm *atol* variable
LSQR_BTOL	real	zero or greater	LSQR algorithm *btol* variable
LSQR_CONLIM	real	zero or greater	LSQR algorithm *conlim* variable
LSQR_ITNLIM	integer	greater than zero	LSQR algorithm *itnlim* variable
LSQR_WRITE	integer	zero or one	instructs PEST to write LSQR file

Table A1.6 Variables in the optional "SVD-assist" section of the PEST control file.

Variable	Type	Values	Description
BASEPESTFILE	text	a filename	name of base PEST control file
BASEJACFILE	text	a filename	name of base PEST Jacobian matrix file
SVDA_MULBPA	integer	zero or one	instructs PEST to record multiple BPA files
SVDA_SCALADJ	integer	-4 to 4	sets type of parameter scaling undertaken in super parameter definition
SVDA_EXTSUPER	integer	0, 1, 2, -2, 3	sets means by which super parameters are calculated
SVDA_SUPDERCALC	integer	zero or one	instructs PEST to compute super parameter sensitivities from base parameter sensitivities
SVDA_PAR_EXCL	integer	0, 1 or -1	if set to 1, instructs PEST to compute super parameters on basis only of observation group in base parameter PEST control file to which pareto-adjustable weighting is assigned in super parameter PEST control file. If set to -1 all groups other than this form basis for super parameter definition

Table A1.7 Variables required for each parameter group in the "parameter groups" section of the PEST control file.

Variable	Type	Values	Description
PARGPNME	text	12 characters or less	parameter group name
INCTYP	text	"relative", "absolute", "rel_to_max"	method by which parameter increments are calculated
DERINC	real	greater than zero	absolute or relative parameter increment
DERINCLB	real	zero or greater	absolute lower bound of relative parameter increment
FORCEN	text	"switch", "always_2", "always_3", "switch_5", "always_5"	determines whether higher order derivatives calculation is undertaken
DERINCMUL	real	greater than zero	derivative increment multiplier when undertaking higher order derivatives calculation
DERMTHD	text	"parabolic", "outside_pts", "best_fit", "minvar", "maxprec"	method of higher order derivatives calculation
SPLITTHRESH	real	greater than zero (or zero to deactivate)	slope threshold for split slope analysis
SPLITRELDIFF	real	greater than zero	relative slope difference threshold for action
SPLITACTION	text	text	"smaller", "zero" or "previous"

Table A1.8 Variables required for each parameter in the "parameter data" section of the PEST control file.

Variable	Type	Values	Description
PARNME	text	12 characters or less	parameter name
PARTRANS	text	"log", "none", "fixed", "tied"	parameter transformation
PARCHGLIM	text	"relative", "factor", or absolute(*N*)	type of parameter change limit
PARVAL1	real	any real number	initial parameter value
PARLBND	real	less than or equal to PARVAL1	parameter lower bound
PARUBND	real	greater than or equal to PARVAL1	parameter upper bound
PARGP	text	12 characters or less	parameter group name
SCALE	real	any number other than zero	multiplication factor for parameter
OFFSET	real	any number	number to add to parameter
DERCOM	integer	zero or greater	model command line used in computing parameter increments
PARTIED	text	12 characters or less	the name of the parameter to which another parameter is tied

Table A1.9 Variables required for each observation group in the "observation groups" section of the PEST control file.

Variable	Type	Values	Description
OBGNME	text	12 characters or less	observation group name
GTARG	real	positive	group-specific target measurement objective function
COVFILE	text	a filename	covariance matrix file associated with group

Table A1.10 Variables required for each observation in the "observation data" section of the PEST control file.

Variable	Type	Values	Description
OBSNME	text	20 characters or less	observation name
OBSVAL	real	any number	measured value of observation
WEIGHT	real	zero or greater	observation weight
OBGNME	text	12 characters or less	observation group to which observation belongs

Table A1.11 Variables in the optional "derivatives command line" section of the PEST control file.

Variable	Type	Values	Description
DERCOMLINE	text	system command	command to run model for derivatives calculation
EXTDERFLE	text	a filename	name of external derivatives file

Table A1.12 Variables in the "model command line" section of the PEST control file.

Variable	Type	Values	Description
COMLINE	text	system command	command to run model

Table A1.13 Variables in the "model input/output" section of the PEST control file.

Variable	Type	Values	Description
TEMPFLE	text	a filename	template file
INFLE	text	a filename	model input file
INSFLE	text	a filename	instruction file
OUTFLE	text	a filename	model output file

Table A1.14 Variables in the optional "prior information" section of the PEST control file.

Variable	Type	Values	Description
PILBL	text	20 characters or less	name of prior information equation
PIFAC	text	real number other than zero	parameter value factor
PARNME	text	12 characters or less	parameter name
PIVAL	real	any number	"observed value" of prior information equation
WEIGHT	real	zero or greater	prior information weight
OBGNME	text	12 characters or less	observation group name

Table A1.15 Variables in the optional "predictive analysis" section of the PEST control file.

Variable	Type	Values	Description
NPREDMAXMIN	integer	-1 or 1	maximise or minimise prediction
PREDNOISE	integer	0 or 1	instructs PEST to include predictive noise in prediction
PD0	real	greater than zero	target objective function
PD1	real	greater than PD0	acceptable objective function
PD2	real	greater than PD1	objective function at which Marquardt lambda testing procedure is altered as prediction is maximised/minimised
ABSPREDLAM	real	zero or greater	absolute prediction change to terminate Marquardt lambda testing
RELPREDLAM	real	zero or greater	relative prediction change to terminate Marquardt lambda testing
INITSCHFAC	real	greater than zero	initial line search factor
MULSCHFAC	real	greater than one	factor by which line search factors are increased along line
NSEARCH	integer	greater than zero	maximum number of model runs in line search
ABSPREDSWH	real	zero or greater	absolute prediction change at which to use central derivatives calculation
RELPREDSWH	real	zero or greater	relative prediction change at which to use central derivatives calculation
NPREDNORED	integer	one or greater	iterations since prediction raised/lowered at which termination is triggered
ABSPREDSTP	real	zero or greater	absolute prediction change at which to trigger termination
RELPREDSTP	real	zero or greater	relative prediction change at which to trigger termination
NPREDSTP	integer	two or greater	number of iterations over which ABSPREDSTP and RELPREDSTP apply

Table A1.16 Variables in the optional "regularisation" section of the PEST control file.

Variable	Type	Values	Description
PHIMLIM	real	greater than zero	target measurement objective function
PHIMACCEPT	real	greater than PHIMLIM	acceptable measurement objective function
FRACPHIM	real	zero or greater, but less than one	set target measurement objective function at this fraction of current measurement objective function
MEMSAVE	text	"memsave" or "nomemsave"	activate conservation of memory at cost of execution speed and quantity of model output
WFINIT	real	greater than zero	initial regularisation weight factor
WFMIN	real	greater than zero	minimum regularisation weight factor
WFMAX	real	greater than WFMAX	maximum regularisation weight factor
LINREG	text	"linreg" or "nonlinreg"	informs PEST that all regularisation constraints are linear
REGCONTINUE	text	"continue" or "nocontinue"	instructs PEST to continue minimising regularisation objective function even if measurement objective function less than PHIMLIM
WFFAC	real	greater than one	regularisation weight factor adjustment factor
WFTOL	real	greater than zero	convergence criterion for regularisation weight factor
IREGADJ	integer	0, 1, 2, 3, 4 or 5	instructs PEST to perform inter-regularisation group weight factor adjustment, or to compute new relative weights for regularisation observations and prior information equations
NOPTREGADJ	integer	one or greater	the optimisation iteration interval for re-calculation of regularisation weights if IREGADJ is 4 or 5
REGWEIGHTRAT	real	absolute value of one or greater	the ratio of highest to lowest regularisation weight; spread is logarithmic with null space projection if set negative
REGSINGTHRESH	real	less than one and greater than zero	singular value of $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ (as factor of highest singular value) at which use of higher regularisation weights commences if IREGADJ is set to 5

Table A1.17 Variables in the optional "pareto" section of the PEST control file.

Variable	Type	Values	Description
PARETO_OBSGROUP	text	12 characters or less	name of observation group whose weights are subject to multiplication by a variable weight factor
PARETO_WTFAC_START	real	zero or greater	initial weight factor for user-specified observation group
PARETO_WTFAC_FIN	real	greater than PARETO_WTFAC_START	final weight factor for user-specified observation group
NUM_WTFAC_INT	integer	greater than zero	number of weight factor increments to employ in traversing Pareto front
NUM_ITER_START	integer	zero or greater	number of optimisation iterations to employ when using initial weight factor
NUM_ITER_GEN	integer	greater than zero	number of optimisation iterations to employ when using any weight factor other than PARETO_WTFAC_START or PARETO_WTFAC_FIN
NUM_ITER_FIN	integer	zero or greater	number of optimisation iterations to employ when using final weight factor
ALT_TERM	integer	zero or one	set to one in order to activate PEST termination determined by value of a specified model output
OBS_TERM	text	20 characters or less	the name of an observation cited in the "observation data" section of the PEST control file whose value will be monitored for possible PEST run termination
ABOVE_OR_BELOW	text	"above" or "below"	determines whether the monitored model output must be above or below the threshold to precipitate run termination
OBS_THRESH	real	any number	value that monitored model output must exceed or undercut to precipitate model run termination
ITER_THRESH	integer	zero or greater	the number of optimisation iterations for which the model output threshold must be exceeded or undercut to precipitate run termination
NOBS_REPORT	integer	zero or greater	number of model outputs whose values to report
OBS_REPORT_N	text	20 characters or less	the name of the *N*'th observation whose value is reported in the POD and PPD files written by PEST when run in "pareto" mode

Appendix B. Files Used by PEST

B1.1 General

In the following tables it is assumed that the name of the PEST control file is *case.pst*. The "*.pst*" filename extension for a PEST control file is mandatory; the filename base must be chosen by the user.

Note that files other than those listed in this appendix are read and written by different members of the PEST utility suite. These are discussed in documentation of those utilities in part II of this manual.

B1.2 PEST Input Files

The files listed in table B1.1 are read by PEST. Some, such as the PEST control file and at least one template and instruction file, are mandatory. Others are not.

Table B1.1 PEST input files.

Name	Function
case.pst	The PEST control file; the "*.pst*" extension is mandatory.
arbitrary	Template file. The filename base and extension are user-selectable; an extension of "*.tpl*" is recommended.
arbitrary	Instruction file. The filename base and extension are user-selectable; an extension of "*.ins*" is recommended.
arbitrary	Observation or prior information covariance matrix file. The filename base and extension are user-selectable.
case.rmf	Run management file required by Parallel PEST and optionally BEOPEST.
case.hld	Parameter hold file employed in user intervention.

B1.3 PEST Output Files

The files listed in the following tables are written by PEST. Not all of these files are written on every run, as some are specific to PEST's mode of operation. The file types are subdivided according to function.

Files in the following tables are not listed in alphabetical order of extension. Rather they are listed in the order of general interest to a PEST user, with related files listed in proximity to each other.

Table B1.1 Information and run record files.

Name	Function
case.rec	Run record file. This is available throughout a PEST run; at the end of a PEST run a suite of run outcome data is added to the file.
case.par	Parameter value file. This file holds best parameter values achieved up until any stage of the inversion process. This file is available throughout a PEST run.
case.par.N	Parameters achieved at the end of iteration N. This file is only saved if PEST is run in "pareto" mode or if the PARSAVEITN control variable is set accordingly.
case.par.N_M	This file is only recorded by BEOPEST. It records parameter values sent to individual slaves. It is saved only if PARSAVERUN is set to "parsaverun".
basecase.bpa	Parameter value file written by PEST when implementing SVD-assisted inversion.
basecase.bpa.N	Iteration-specific parameter value file written by PEST when implementing SVD-assisted inversion.
###error.par.N	A parameter value file recording parameters used by the model on the *N*'th occasion of model run failure encountered when testing parameter upgrades calculated using different values of the Marquardt lambda. These files are written only if the LAMFORGIVE control variable is set to "lamforgive".
case.jco	The binary Jacobian matrix file. This is available on a continuous basis; it is updated whenever parameters are improved. To obtain an ASCII version use the JACWRIT utility.
case.jco.N	Iteration-specific Jacobian matrix files which are saved if the JCOSAVEITN control variable is set to "jcosaveitn".
case.sen	This file contains composite parameter sensitivities; it is updated during every iteration.
case.seo	Contains composite observation sensitivities. It is written at the end of a PEST run, unless PEST is run in "pareto" mode.
case.res	Residuals file. This lists measured and model-calculated values of all observations and prior information equations, together with residuals, weighted residuals, and residuals processed in other ways. It is written on completion of a PEST run unless PEST is run in "pareto" mode.

case.rsr	Rotated residuals file. This file is written only if one or more observation covariances matrices are provided. It is equivalent to the residuals file, but records information pertaining to covariance-matrix-transformed residuals. It is not recorded if PEST is run in “pareto” mode.
case.rei	Interim residuals file. This file contains the same information as the residuals file, but is available during progression of the inversion process, and updated during every iteration in which parameters improve. It is not recorded if PEST is run in “pareto” mode.
case.rei.N	An interim residuals file containing residuals computed using parameters estimated during iteration *N* of the inversion process. This file can be requested through the REISAVEITN variable.
case.mtt	The matrix file holds covariance, correlation and eigencomponent matrices if no regularisation is employed. It is updated at the end of very iteration. At the end of the inversion process, matrix data pertaining to optimised parameters is recorded in the same format.
case.rsd	The resolution data file. This is a binary file which is used by utilities such as RESPROC and RESWRIT for computation of post-calibration error variance.

Table B1.2 Files specific to inverse problem solution method.

Name	Function
case.cnd	The condition number file. This records the condition number of the "normal matrix" $\mathbf{J}^t\mathbf{QJ}$. It is not recorded if either SVD or LSQR is used for solution of the inverse problem. The file is updated on each occasion that a new parameter upgrade vector is calculated using a new Marquardt lambda.
case.svd	This file records singular values and, optionally, associated columns of the $\mathbf{V}_1$ matrix. It is recorded only if singular value decomposition, either of $\mathbf{Q}^{1/2}\mathbf{J}$ or $\mathbf{J}^t\mathbf{QJ}$ is employed in solution of the inverse problem.
case.lsq	This file records usage and solution details of the LSQR algorithm. If is recorded if the LSQRWRITE variable is set to 1 and LSQR is used in solution of the inverse problem. It is updated on each occasion that a new parameter upgrade vector is calculated.

TableB1.3 Files employed when PEST is undertaking SVD-assisted inversion.

Name	Function
parcalc.in	Input file for the PARCALC utility which transforms super parameters to base parameters.
parcalc.tpl	Template file for *parcalc.in*.
picalc.in	Input file for the PICALC utility which evaluates prior information equations which cite base parameters.
picalc.tpl	Template file for *picalc.in*.
picalc.out	File written by the PICALC utility.
picalc.ins	Instruction file used to read *picalc.out*.
svdabatch.bat	Batch file used by PEST as "the model".

Table B1.4 Files written when PEST is run in "pareto" mode.

Name	Function
case.par.N	Parameters at end of iteration *N*.
case.ppd	Pareto parameter data file. This binary file is updated during ever PEST iteration. Its contents can be examined using the PPD2ASC and PPD2PAR utilities.
case.pod	Pareto objective function file. This is recorded continuously as PEST runs in "pareto" mode.

Table B1.5 Files used to store restart information (all binary files).

Name	Function
case.rst	Contains restart information pertaining to the present iteration.
case.jst	Contains restart information pertaining to the previous iteration.
case.jac	Contains jacobian matrix pertaining to previous iteration.
case.drf	Contains semi-complete Jacobian matrix for present iteration.
case.prf	Contains semi-complete contents of model output run queue for present iteration.

Table B1.6 Files used in parallel run management.

Name	Function
case.rmf	The Parallel PEST run management file. Optionally, this file can also be used to inform BEOPEST of the value of the PARLAM variable.
case.rmr	Parallel PEST and BEOPEST run management record file.
pest.rdy	Message file from Parallel PEST to PSLAVE.
param.rdy	Message file from Parallel PEST to PSLAVE.
pslave.fin	Message file from Parallel PEST to PSLAVE.
pslave.rdy	Message file from PSLAVE to Parallel PEST.
observ.rdy	Message file from PSLAVE to Parallel PEST.
p###.##	Used by PEST to test communication with PSLAVE working directories.
pest###.dap	Direct access binary file used as parameter value parallel run queue. This is used by both BEOPEST and Parallel PEST.
pest###.dao	Direct access binary file used to store model output values provided by slaves. This is used by both BEOPEST and Parallel PEST.