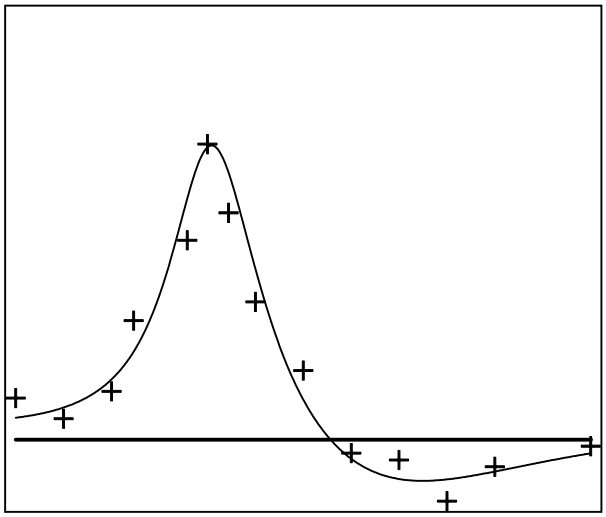
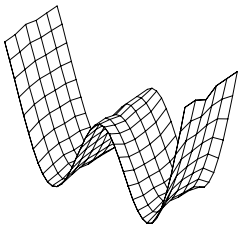
PEST

Model-Independent Parameter Estimation





Watermark Numerical Computing

Acknowledgments

Some of the improvements made to PEST documented in this manual were carried out while I was employed as a Research Scientist at the University of Idaho, Idaho Falls. Also, while I occupied that position, I was able to devote a considerable amount of time to writing software to expedite PEST's usage in the calibration and predictive analysis of surface water models. Much of this software is now part of the PEST Surface Water Utilities Suite.

I wish to publicly acknowledge my gratitude to the University of Idaho for providing me with the time and resources necessary to carry out this important work.

Disclaimer

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

The author does not make any expressed of 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.

Preface to First Edition

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

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 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 optimisation 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

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 and 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 defined as the magnitude of the vector comprising the column of the Jacobian matrix pertaining to that parameter divided by the number of observations. 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 x least sensitive parameters within a certain group be held while the parameter upgrade vector is calculated, where x 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-calculated 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 the Third Edition

Production of the third edition of the PEST manual coincides with the release of Version 4.01 of PEST, also known as PEST2000. The principal addition to PEST functionality encapsulated in PEST2000 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 most other methods, the PEST predictive analysis algorithm relies on no assumptions concerning the linearity of a model or the probability distribution associated with its parameters;

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 any other method of nonlinear predictive 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 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, otherwise known as PEST-ASP ("ASP" stands for "Advanced Spatial Parameterisation"). 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 PEST-ASP, 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 Ground Water Data 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 PEST-ASP 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 (viz. 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 proceeds. A new utility program named READJAC 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

Bugs

In the unlikely event that you discover a bug in PEST, please report it to me, John Doherty, at the following email address:-

jdoherty@gil.com.au

Table of Contents

1. Introduction

1.1 Installation

Installation instructions are provided on the printed sheet accompanying this manual; follow these instructions to transfer PEST executable and support files to your machine's hard disk.

Your *autoexec.bat* should be modified before you run PEST; the PEST directory must be added to the PATH statement.

1.2 The PEST Concept

1.2.1 A Model's Data Requirements

There is a mathematical model for just about everything. Computer programs have been written to describe the flow of water in channels, the flow of electricity in conductors of strange shape, the growth of plants, the population dynamics of ants, the distribution of stress in the hulls of ships and on and on. Modelling programs generally require data of four main types. These are:

- **Fixed data.** These data define the system; for example in a ground water model the shape of the aquifer is fixed, as are the whereabouts of any extraction and injection bores.

- **Parameters.** These are the properties of the system; parameters for a ground water model include the hydraulic conductivity and storage capacity of the rocks through which the water flows, while for a stress model parameters include the elastic constants of the component materials. A model may have many parameters, each pertaining to one particular attribute of the system which affects its response to an input or excitation. In spatial models a particular system property may vary from place to place; hence the parameter data needed by the model may include either individual instances of that property for certain model subregions, or some numbers which describe the manner in which the property is spatially distributed.

- **Excitations**. These are the quantities which "drive" the system, for example climatic data in a plant growth model, and the source and location of electric current in electromagnetic boundary-value problems. Like parameters, excitations may have spatial dependence.

- **Control data**. These data provide settings for the numerical solution method by which the system equations are solved. Examples are the specifications of a finite element mesh, the convergence criteria for a preconditioned conjugate gradient matrix equation solver, and so on.

The distinction between these different data types may not always be clear in a particular case.

The purpose of a mathematical model is to produce numbers. These numbers are the model's predictions of what a natural or man-made system will do under a certain excitation regime. It is for the sake of these numbers that the model was built, be it a ten line program involving a few additions and subtractions, or a complex numerical procedure for the solution of coupled sets of nonlinear partial differential equations.

Where a model simulates reality it often happens that the model-user does not know what reality is; in fact models are often used to infer reality by comparing the numbers that they produce with numbers obtained from some kind of measurement. Thus if a model's parameter and/or excitation data are "tweaked", or adjusted, until the model produces numbers that compare well with those yielded by measurement, then perhaps it can be assumed that the excitations or parameters so obtained have actually told us something which we could not obtain by direct observation. Thus if a ground water model is able to reproduce the variations in borehole water levels over time (a quantity which can be obtained by direct observation), the hydraulic conductivity values that we assign to different parts of the model domain in order to achieve this match may be correct; this is fortunate as it is often difficult or expensive to measure rock hydraulic conductivities directly. Similarly, if the resistivities and thicknesses that we assign to a layered half-space reproduce voltage/current ratios measured at various electrode spacings on the surface of the half-space, then perhaps these resistivities and thicknesses represent a facet of reality that it may not have been possible to obtain by direct observation.

1.2.2 The Role of PEST

PEST is all about using existing models to infer aspects of reality that may not be amenable to direct measurement. In general its uses fall into three broad categories. These are:

- **Interpretation**. In this case an experiment is set up specifically to infer some property of a system, often by disturbing or exiting it in some way. A model is used to relate the excitations and system properties to quantities that can actually be measured. An interpretation method may then be based on the premise that if the excitation is known it may be possible to estimate the system properties from the measurement set. Alternatively, if system properties are known it may be possible to use the model to infer something about the excitation by adjusting model input excitation variables until model outcomes match measurements. (In some cases it may even be possible to estimate both excitations and parameters.) A good deal of geophysical software falls into this category, where sometimes very elegant mathematical models are developed in order to infer aspects of the earth's structure from measurements that are confined either to the earth's surface or to a handful of boreholes.

- **Calibration**. If a natural or man-made system is subject to certain excitations, and numbers representing these same excitations are supplied to a model for that system, it may be possible to adjust the model's parameters until the numbers which it generates correspond well with certain measurements made of the system which it simulates. If so, it may then be possible to conclude that the model will represent the system's behaviour adequately as the latter responds to other excitations as well - excitations which we may not be prepared to give the system in practice. A model is said to be "calibrated" when its parameters have been adjusted in this fashion.

- **Predictive Analysis**. Once a parameter set has been determined for which model behaviour matches system behaviour as well as possible, it is then reasonable to ask whether another parameter set exists which also results in reasonable simulation by the model of the system under study. If this is the case, an even more pertinent question is whether predictions made by the model with the new parameter set are different. Depending on the system under study and the type of model being used to study this system, the ramifications of such differences may be extremely important.

The purpose of PEST (which is an acronym for Parameter ESTimation) is to assist in data interpretation, model calibration and predictive analysis. Where model parameters and/or excitations need to be adjusted until model-generated numbers fit a set of observations as closely as possible then, provided certain continuity conditions are met (see the next section), PEST should be able to do the job. PEST will adjust model parameters and/or excitations until the fit between model outputs and laboratory or field observations is optimised in the weighted least squares sense. Where parameter values inferred through this process are nonunique, PEST will analyse the repercussions of this nonuniqueness on predictions made by the model. The universal applicability of PEST lies in its ability to perform these tasks for *any* model that reads its input data from one or a number of ASCII (ie. text) input files and writes the outcomes of its calculations to one or more ASCII output files. Thus a model does not have to be recast as a subroutine and recompiled before it can be used within a parameter estimation process. *PEST adapts to the model, the model does not need to adapt to PEST*.

Thus PEST, as a nonlinear parameter estimator, can exist independently of any particular model, yet can be used to estimate parameters and/or excitations, and carry out various predictive analysis tasks, for a wide range of model types. Thus PEST can turn just about any existing computer model, be it a home-made model based on an analytical solution to a simple physical problem, a semi-empirical description of some natural process, or a sophisticated numerical solver for a complex boundary-value problem, into a powerful nonlinear parameter estimation package for the system which that model simulates.

1.3 What Pest Does

Models produce numbers. If there are field or laboratory measurements corresponding to some of these numbers, PEST can adjust model parameter and/or excitation data in order that the discrepancies between the pertinent model-generated numbers and the corresponding measurements are reduced to a minimum. It does this by taking control of the model and running it as many times as is necessary in order to determine this optimal set of parameters and/or excitations. You, as the model user, must inform PEST of where the adjustable parameters and excitations are to be found on the model input files. Once PEST is provided with this information, it can rewrite these model input files using whatever parameters and excitations are appropriate at any stage of the optimisation process. You must also teach PEST how to identify those numbers on the model output files that correspond to observations that you have made of the real world. Thus, each time it runs the model, PEST is able to read those model outcomes which must be matched to field or laboratory observations. After calculating the mismatch between the two sets of numbers, and evaluating how best to correct that mismatch, it adjusts model input data and runs the model again.

For PEST to take control of an existing model in this fashion in order to optimise its parameters and/or excitations, certain conditions must be met. These are as follows:

- While a model may read many input files, some of which may be binary and some of which may be ASCII, the file or files containing those excitations and/or parameters which PEST is required to adjust must be ASCII (ie. text) files.

- While a model may write many output files, some of which may be binary and some of which may be ASCII, the file or files containing those model outcomes which complement field or laboratory measurements must be ASCII (ie. text) files.

- The model must be capable of being run using a system command, and of requiring no user intervention to run to completion (see below for further details).

- PEST uses a nonlinear estimation technique known as the Gauss-Marquardt-Levenberg method. The strength of this method lies in the fact that it can generally estimate parameters using fewer model runs than any other estimation method, a definite bonus for large models whose run times may be considerable. However the method requires that the dependence of model-generated observation counterparts on adjustable parameters and/or excitations be continuously differentiable.

PEST must be provided with a set of input files containing the data which it needs in order to effectively take control of a particular model. Specifications for these files will be described later in the manual; their preparation is a relatively simple task. Amongst the data which must be supplied to PEST is the name of the model of which it must take control. In the simplest case, this may be the name of a single executable file, ie. a program that simulates a system for which parameterisation or excitation estimation is required. In more complex cases the name may pertain to a batch program which runs a number of executable programs in succession. Thus output data from one model can feed another, or a model postprocessor may extract pertinent model outputs from a lengthy binary file and place them into a smaller, "tidy" file in ASCII format for easy PEST access.

Models which receive their data directly from the user through keyboard entry and write their results directly to the screen can also be used with PEST. Keyboard inputs can be typed ahead of time into a file, and the model directed to look to this file for its input data using the "<" symbol on the model command line; likewise model screen output can be redirected to a file using the ">" symbol. PEST can then be instructed to alter parameters and/or excitations on the input file and read numbers matching observations from the output file. Thus it can run the model as many times as it needs to without any human intervention.

PEST can be used with models written in any programming language; they can be home-made or bought. You do not need to have the source code or know much about the internal workings of the model. Models can be small and fast, finishing execution in the blink of an eye, or they can be large and slow, taking minutes or even hours to run; it does not matter to PEST.

1.4 An Overview of PEST

PEST can be subdivided into three functionally separate components whose roles are:

- parameter and/or excitation definition and recognition,
- observation definition and recognition, and
- the nonlinear estimation and predictive analysis algorithm.

Though the workings of PEST will be described in detail in later chapters, these three components are discussed briefly so that you can become acquainted with PEST's capabilities.

1.4.1 Parameter Definition and Recognition

From this point on, the single word "parameter" is used to describe what has hitherto been referred to as "parameters and/or excitations".

Of the masses of data of all types that may reside on a model's input files, those numbers must be identified which PEST is free to alter and optimise. Fortunately, this is a simple process which can be carried out using input file "templates". If a model requires, for example, five input files, and two of these contain parameters which PEST is free to adjust, then a template file must be prepared for each of the two input files containing adjustable parameters. To construct a template file, simply start with a model input file and replace each space occupied by a parameter by a set of characters that both identify the parameter and define its width on the input file. Then whenever PEST runs the model it copies the template to the model input file, replacing each parameter space with a parameter value as it does so.

PEST template files can be constructed from model input files using any text editor. They can be checked for syntactical correctness and consistency using the utility programs PESTCHEK and TEMPCHEK.

An important point to note about template files is that a given parameter (identified by a unique name of up to twelve characters in length) can be referenced once or many times. The fact that it can be referenced many times may be very useful when working with large numerical models. For example, a finite-difference model may be used to calculate the electromagnetic fields within a half-space, the half-space being subdivided into a number of zones of constant electrical conductivity. The model may need to be supplied with a large two (or even three) dimensional array in which conductivity values are disposed in a manner analogous to their disposition in the half-space. Each half-space zone is defined by that part of the array containing elements of a particular value, this value providing the conductivity pertaining to that zone. It may be these zone conductivity values that we wish to optimise. Fortunately, creating a template for the model input file holding the array is a simple matter, for each occurrence of a particular zone-defining number in the original input file can be replaced by the parameter identifier specific to that zone. Hence every time PEST rewrites the array, all array elements belonging to a certain zone will have the same value, specific to that zone.

On a particular PEST run a parameter can remain fixed if desired. Thus, while the parameter may be identified on a template file, PEST will not adjust its value from that which you supply at the beginning of the parameter estimation process. Another feature is that one or a number of parameters can be "tied" to a "parent" parameter. In this case, though all such tied

parameters are identified on template files, only the parent parameter is actually optimised; the tied parameters are simply varied with this parameter, maintaining a constant ratio to it.

PEST requires that upper and lower bounds be supplied for adjustable parameters (ie. parameters which are neither fixed not tied); this information is vital to PEST, for it informs PEST of the range of permissible values that a parameter can take. Many models produce nonsensical results, or may incur a run-time error, if certain inputs transgress permissible domains. For example, parameters such as electrical conductivity and solute concentration should never be provided with negative values. Also, if a parameter occurs as a divisor anywhere in the model's code, it can never be zero.

For many models it has been found that if the logarithms of certain parameters are optimised, rather than the parameters themselves, the rate of convergence to optimal parameter values can be considerably hastened; PEST allows such logarithmic transformation of selected parameters.

Often there is some information available from outside of the parameter estimation process about what value a parameter should take. Alternatively, you may know that the sum or difference of two or more parameters (or their product or quotient in the case of logarithmically-transformed parameters) should assume a certain value. PEST allows you to incorporate such prior information into the estimation process by increasing the value of the objective function (ie. the sum of squared deviations between model and observations - see the next section) in proportion to the extent to which these articles of prior information are transgressed.

Finally, parameters adjusted by PEST can be scaled and offset with respect to the parameters actually used by the model. Thus you may wish to subtract 273.15 from an absolute temperature before writing that temperature to a model input file which requires Celcius degrees; or you may wish to negate a model parameter which never becomes positive so that it can be log-transformed by PEST for greater optimisation efficiency.

1.4.2 Observation Definition and Recognition

From this point onwards, those numbers on a model output file for which there are corresponding "real-world" values to which they must be matched will be referred to simply as "observations". Of the masses of data produced by a model, only a handful of numbers may actually be "observations". For example, a population dynamics model may calculate population figures on a daily basis, yet measurements may only have been taken every week. In this case most of the model's output data will be redundant from the point of view of model calibration. Similarly, a model for the stress field surrounding an excavation may calculate stress figures for each of the thousands of nodes of a finite-element mesh, through the use of which the system differential equations are solved; however stress measurements may be available at only a handful of points, viz. at the locations of specially-installed stress sensors in the rocks surrounding the excavation. Again PEST must be able to identify a handful of numbers (viz. stress values calculated at those points for which stress measurements are available) out of the thousands that may be written to the model's output file.

In order to peruse a model output file and read the observation values calculated by the

model, PEST must be provided with a set of instructions. Unfortunately, the template concept used for model input files will not work for model output files as the latter may change from run to run, depending on parameter values. However, if a person is capable of locating a pertinent model output amongst the other data on a model output file, then so too is a computer. As it turns out, the instruction set by which this can be achieved is relatively simple, involving only a handful of basic directives.

PEST requires, then, that for each model output file which must be opened and perused for observation values, an instruction file be provided detailing how to find those observations. This instruction file can be prepared using any text editor. It can be checked for syntactical correctness and consistency using the utility programs PESTCHEK and INSCHEK.

Once interfaced with a model, PEST's role is to minimise the weighted sum of squared differences between model-generated observation values and those actually measured in the laboratory or field; this sum of weighted, squared, model-to-measurement discrepancies is referred to as the "objective function". The fact that these discrepancies can be weighted makes some observations more important than others in determining the optimisation outcome. Weights should be inversely proportional to the standard deviations of observations, "trustworthy" observations having a greater weight than those which cannot be trusted as much. Also, if observations are of different types (for example solute concentration and solvent flow rates in a chemical process model) the weights assigned to the two observation types should reflect the relative magnitudes of the numbers used to express the two quantities; in this way the set of larger numbers will not dominate the parameter estimation process just because the numbers are large. A particular observation can be provided with a weight of zero if you do not wish it to affect the optimisation process at all.

Like parameters, you must provide each observation with a name of up to twelve characters in length; PEST uses this name to provide you with information about that observation.

1.4.3 The Parameter Estimation Algorithm

The Gauss-Marquardt-Levenberg algorithm used by PEST is described in detail in the next chapter. For linear models (ie. models for which observations are calculated from parameters through a matrix equation with constant parameter coefficients), optimisation can be achieved in one step. However for nonlinear problems (most models fall into this category), parameter estimation is an iterative process. At the beginning of each iteration the relationship between model parameters and model-generated observations is linearised by formulating it as a Taylor expansion about the currently best parameter set; hence the derivatives of all observations with respect to all parameters must be calculated. This linearised problem is then solved for a better parameter set, and the new parameters tested by running the model again. By comparing parameter changes and objective function improvement achieved through the current iteration with those achieved in previous iterations, PEST can tell whether it is worth undertaking another optimisation iteration; if so the whole process is repeated.

At the beginning of a PEST run you must supply a set of initial parameter values; these are the values that PEST uses at the start of its first optimisation iteration. For many problems only five or six optimisation iterations will be required for model calibration or data interpretation. In other cases convergence will be slow, requiring many more optimisation

iterations. Often a proper choice of whether and what parameters should be logarithmically transformed has a pronounced effect on optimisation efficiency; the transformation of some parameters may turn a highly nonlinear problem into a reasonably linear one.

Derivatives of observations with respect to parameters are calculated using finite differences. During every optimisation iteration the model is run once for each adjustable parameter, a small user-supplied increment being added to the parameter value prior to the run. The resulting observation changes are divided by this increment in order to calculate their derivatives with respect to the parameter. This is repeated for each parameter. This technique of derivatives calculation is referred to as the method of "forward differences".

Derivatives calculated in this way are only approximate. If the increment is too large the approximation will be poor; if the increment is too small roundoff errors will detract from derivatives accuracy. Both of these effects will degrade optimisation performance. To combat the problem of derivatives inaccuracy, PEST allows derivatives to be calculated using the method of "central differences". Using this method, two model runs are required to calculate a set of observation derivatives with respect to any parameter. For the first run an increment is added to the current parameter value, while for the second run the increment is subtracted. Hence three observation-parameter pairs are used in the calculation of any derivative (the third pair being the current parameter value and corresponding observation value). The derivative is calculated either by (i) fitting a parabola to all three points, (ii) constructing a best-fit straight line for the three points or (iii) by simply using finite differences on the outer two points (its your choice).

It is normally best to commence an optimisation run using the more economical forward difference method, allowing PEST to switch to central differences when the going gets tough. PEST will make the switch automatically according to a criterion which you supply to it prior to the commencement of the run.

PEST's implementation of the Gauss-Marquardt-Levenberg method is extremely flexible; many aspects of it can be varied to suite the problem at hand, allowing you to optimise PEST's performance for your particular model. How you do this is described later in this manual. In the course of the estimation process PEST writes what it is doing to the screen; it simultaneously writes a more detailed run record to a file. You can pause PEST execution at any time to inspect its outputs in detail; when you have finished looking at these, PEST will recommence execution exactly where it was interrupted. Alternatively, you can shut down PEST completely at any stage. It can then be restarted later; you can direct it to recommence execution either at the beginning of the optimisation iteration in which it was interrupted or at that point within the current or previous iteration at which it last attempted to upgrade parameter values.

As it calculates derivatives, PEST records the sensitivity of each parameter with respect to the observation dataset to a file which is continuously available for inspection. If it is judged that PEST's performance is being inhibited by the behaviour of certain parameters (normally the most insensitive ones) during the optimisation process, these parameters can be temporarily held at their current values while PEST calculates a suitable upgrade for the rest of the parameters. If desired, PEST can be requested to repeat its determination of the parameter upgrade vector with further parameters held fixed. Certain variables governing the operation of the Gauss-Marquardt-Levenberg method in determining the optimum upgrade

vector can also be adjusted prior to repeating the calculation. Thus you can interact with PEST, assisting it in its determination of optimum parameter values in difficult situations if you so desire.

At the end of the parameter estimation process (the end being determined either by PEST or by you) PEST writes a large amount of useful data to its run record file. PEST records the optimised value of each adjustable parameter together with that parameter's 95% confidence interval. It tabulates the set of field measurements, their optimised model-calculated counterparts, the difference between each pair, and certain functions of these differences. (These are also recorded on a special file ready for immediate importation into a spreadsheet for further processing.) Then it calculates and prints/displays three matrices, viz. the parameter covariance matrix, the parameter correlation coefficient matrix and the matrix of normalised eigenvectors of the covariance matrix.

1.4.4 Predictive Analysis

When used to calibrate a model (the traditional use of PEST), PEST is asked to minimise an objective function comprised of the sum of weighted squared deviations between certain model outcomes and their corresponding field-measured counterparts. When undertaking this task, PEST is run in "parameter estimation mode".

It is a sad fact of model usage that there are often many different sets of parameter values for which the objective function is at its minimum or almost at its minimum. Thus there are many different sets of parameters which could be considered to calibrate a model. A question that then arises is: "if I use different sets of parameter values when using the model to make predictions (all of these sets being considered to calibrate the model), will I get different values for key model outcomes?". This question can be answered by running PEST in "predictive analysis mode". To run PEST in predictive analysis mode the user informs PEST of the objective function value below which the model can be considered to be calibrated; this value is normally just slightly above the minimum objective function value as determined in a previous PEST calibration run. A key model prediction is then identified on one of the model output files; this may involve setting up a "dual model" (run by PEST through a batch file) consisting of the model run under both calibration and predictive conditions. PEST is then asked to find that parameter set which results in the maximum or minimum model prediction while still calibrating the model. In doing this, PEST uses an iterative solution procedure similar in many ways to that used for solution of the parameter estimation problem. The key model prediction made with a parameter set calculated in this way defines the upper or lower bound of the uncertainty interval associated with that prediction.

Most aspects of PEST usage in predictive analysis mode are identical to PEST's usage in parameter estimation mode. In particular, bounds can be placed on adjustable parameters, one parameter can be tied to another, parameters can be logarithmically transformed for greater problem linearity, troublesome parameters can be temporarily held while the parameter upgrade vector is re-calculated, etc. However the end-point of the iterative solution process is no longer a minimised objective function; it is a maximised or minimised prediction with the objective function being as close as possible to that defining the acceptable limit for model calibration.

Once a key model prediction has been identified, it is also possible to ask another important

question of PEST. The question is, " is it possible to find a parameter set for which the key model prediction is a certain value while still maintaining the calibration objective function at or below the acceptable calibration limit?" This is similar to the question that is answered by running PEST in predictive analysis mode. However it differs slightly from that question in that a maximum or minimum prediction is no longer being sought; rather the acceptability of a certain prediction in terms of the model's ability to satisfy calibration constraints is being tested. Once again a "dual model" is required in which the model is run under both calibration and predictive conditions. PEST answers the question by attempting to minimise a new objective function which incorporates not just the differences between model-generated and observed quantities under calibration conditions, but the difference between the key prediction and the user-specified value of this prediction when the model is run under predictive conditions. Thus PEST is run in its traditional parameter estimation mode with a slightly altered objective function. When run in this manner, PEST's run-time outputs are adjusted such that information on the key model prediction is recorded, together with information on all other aspects of the parameter estimation process.

1.4.5 Regularisation

In its broadest sense, "regularisation" is a term used to describe the process whereby a large number of parameters can be simultaneously estimated without incurring the numerical instability that normally accompanies parameter nonuniqueness. Numerical stability is normally achieved through the provision of "supplementary information" to the parameter estimation process. Such "supplementary information" often takes the form of preferred values for parameters, or for relationships between parameters. Thus if, for a particular parameter, the information content of the observation dataset is such that a unique value cannot be estimated for that parameter on the basis of that dataset alone, uniqueness can nevertheless be achieved by using the supplementary information provided for that parameter through the regularisation process.

A problem that arises when using such supplementary information as part of a traditional parameter estimation exercise is the determination of how much notice should be taken of this information in comparison to the notice taken of the observation dataset against which the model is being calibrated. If the supplementary information is given too much weight in the parameter estimation process the observation dataset may be ignored. On the other hand, if it is given too little weight, the stabilization potential of the supplementary dataset will not be realized. When using PEST in "regularisation mode" PEST takes care of this problem, for PEST calculates the relative weighting given to the two sets of information itself. In this way, the supplementary information is used only to the extent necessary to ensure stability of the parameter estimation process. Or, looked at another way, if the information content of the calibration dataset is insufficient to provide unique estimation of certain parameters, then PEST will automatically elevate the status of the supplementary information such that this provides the grounds for unique estimation of those parameters.

PEST's regularisation functionality is useful in many types of modelling – particularly where many different parameters must be estimated for complex systems. It is particularly useful in estimating values for parameters which describe the spatial distribution of some property over a two- or three-dimensional model domain, for example a ground water or geophysical model. The user is no longer required to subdivide the model domain into a small number of

zones of piecewise parameter constancy. Rather, a large number of parameters can be used to describe the distribution of the spatial property and PEST's regularisation functionality can be used to estimate values for these parameters. If supplementary information is provided in the form of a "preferentially smooth system state", then only enough heterogeneity will be introduced to the system to guarantee a good fit between model outcomes and field data. Furthermore, PEST will determine the locations of areas of anomalous property values itself. This is normally a vastly superior method by which to infer the distribution of a spatial parameter over a model domain than to estimate parameters associated with a pre-defined zonation pattern.

PEST's regularisation functionality can also be useful when calibrating a number of models simultaneously (for example rainfall-runoff models in different watersheds). PEST can be asked to preferentially estimate identical values for the same parameter types in the different model domains. Differences in parameter values estimated through the regularised multi-model calibration process will then be present because they *must* be present if all of the models are to match their corresponding field measurements well.

1.5 How to Use PEST

The PEST suite is comprised of two versions of PEST and six utility programs for building and checking PEST input files. A sensitivity analyser and a parameter preprocessor are also supplied with PEST. All of these programs are command-line driven programs, ie. they can be run from a command-line window by typing the name of the appropriate executable at the screen prompt. Note, however, that they are all true WINDOWS executables.

A suite of utility programs is also available to enhance the use of PEST in certain modelling contexts. See, for example, the PEST Ground Water and Surface Water Modelling Utilities.

1.5.1 The Two Versions of PEST

The two variants of PEST are the "single window" version of PEST and "Parallel PEST".

In the single window version of PEST (which is run through the "pest" command), the model shares the same window as PEST, with the result that screen output generated by the model is interspersed with that generated by PEST (unless the former is re-directed to a *nul* file - see later).

Parallel PEST (which is run through the "ppest" command) is able to run multiple instances of a model in parallel, either in different command-line windows on the same machine, or on different (networked) machines. By undertaking simultaneous model runs, enormous savings in overall optimisation time can be made, particularly when calibrating large and complex models. Preparation for a Parallel PEST run requires the creation of an extra input file; also a slave program (PSLAVE) must be run on each machine on which Parallel PEST runs the model.

PEST execution can be interrupted or stopped at any time. To do this, run one of the programs PPAUSE, PUNPAUSE, PSTOP or PSTOPST to achieve the desired effect.

1.5.2 PEST Utilities

PEST requires three types of input file. These are:

- template files, one for each model input file which PEST must write prior to a model run,
- instruction files, one for each model output file which PEST must read after a model run, and
- a PEST control file which "brings it all together", supplying PEST with the names of all template and instruction files together with the model input/output files to which they pertain. It also provides PEST with the model name, parameter initial estimates, field or laboratory measurements to which model outcomes must be matched, prior parameter information, and a number of PEST variables which control the implementation of the Gauss-Marquardt-Levenberg method.

You must prepare the template and instruction files yourself. This can be easily done using a text editor; full details are provided in Chapter 3 of this manual. After you have prepared a template file, you can use program TEMPCHEK to check that it has no syntax errors. Furthermore, if you supply TEMPCHEK with a set of parameter values, it will write a model input file on the basis of the template file which you have just prepared. You can then run your model, making sure that it reads the input file correctly. In this way you can be sure, prior to running PEST, that PEST will write a model input file that satisfies your model's requirements.

INSCHEK does for instruction files what TEMPCHEK does for template files. INSCHEK checks that an instruction file is syntactically correct and consistent. Then, if you wish, INSCHEK will read a model output file using the directives contained in the instruction file, listing the values of all observations cited in the instruction file as read from the model output file. In this way you can be sure, prior to running PEST, that PEST will read a model output file correctly.

Like template and instruction files, the PEST control file can be prepared using a text editor. However it is generally easier to prepare it using program PESTGEN. PESTGEN generates a PEST control file using parameter and observation names cited in template and instruction files which have already been built. However, as it uses default values for all variables which control PEST execution, you will probably need to make some changes to a PESTGEN-generated PEST control file (usually not too many) in order to tune PEST to your current problem.

After all PEST input files have been prepared (viz. the PEST control file and all template and instruction files) you can use program PESTCHEK to check that the entire PEST input dataset contained in these files is consistent and complete.

Once PEST has been run and an improved parameter set obtained, you may wish to build a new PEST control file using the improved parameter estimates as initial estimates for another run. This may occur if you wish to alter some facet of the model, add prior information, alter a PEST variable or two, etc. prior to continuing with the optimisation process. Program PARREP allows you to replace initial parameter values as recorded on a PEST control file

with those recorded in a "parameter value file", the latter (having been written by PEST) containing the best parameter values achieved on the previous PEST run.

If you wish to generate a file containing the sensitivity of each model output for which there is a corresponding field or laboratory measurement with respect to each adjustable parameter, use the JACWRIT utility program. JACWRIT translates a binary file (written by PEST) containing this useful information into ASCII format for easy user inspection.

Note that because PEST input files are simple text files, for which full construction details are provided in this manual, they can be prepared by other software, for example by a text editor or by a program that you may write yourself in order to automate PEST file generation for a specific application. Thus, if you wish, you can integrate PEST into your modelling suite so that model parameter estimation becomes as straightforward as modelling itself.

1.5.3 Parameter Preprocessing

Sometimes it is useful to undertake complex mathematical operations on model parameters before actually providing them to the model. This can help the parameter estimation process in a number of ways. For example, appropriate parameter transformation may render a model more linear with respect to one or more of its parameters; in other circumstances the calculation of "secondary parameters" (eg. monthly variation of a particular model input type) from a smaller number of "primary parameters" which describe the seasonal variation of the secondary parameter (eg. the mean, amplitude and phase of that parameter's variation) can bring stability to the parameter estimation process by allowing PEST to estimate a fewer number of parameters while incorporating the user's knowledge of the type of variation that the parameter undergoes directly into the parameter estimation process. Parameter transformations of this type (and many more) can be undertaken using the parameter preprocessor PAR2PAR supplied with PEST. PAR2PAR requires a text input file (from which a template file is easily prepared) describing the mathematical relationships (which can be of arbitrary complexity) that exist between parameters. Like PEST, it then writes its "secondary parameters" to a model input file using a template of that file. During the calibration process undertaken by PEST, PAR2PAR is run just before the actual model executable within a batch file comprising the "composite model".

1.5.4 Sensitivity Analysis

SENSAN (which stands for "SENSitivity Analysis") is a command-line program which provides the capability to carry out multiple model runs without user intervention, using different parameter values for each run. Thus a computer can be kept busy all night undertaking successive model runs, with key model outputs from each run being recorded in a format suitable for easy later analysis using a spreadsheet or other data processing package. Any or all model output files for specific model runs can also be stored in their entirety under separate names if desired.

SENSAN uses the same model interface protocol as PEST does, ie. parameter values are supplied to a model through model input file templates, and key model-generated numbers are read from model output files using an instruction set. In addition, a special "SENSAN control file" must be built, informing SENSAN of the names of all template, instruction and model input/output files, the model command line, and the parameter values that must be

used for each model run.

SENSAN is accompanied by a checking program named SENSCHEK. The role and operation of SENSCHEK are very similar to those of PESTCHEK, viz. it checks all SENSAN input data to verify that it is consistent and correct. SENSCHEK reads a SENSAN control file, as well as all template and instruction files cited therein. If any errors or inconsistencies are detected, appropriate messages are written to the screen.

Note that SENSAN and SENSCHEK are both true WINDOWS executables.

1.6 This Manual

This introduction has provided an overview of the capabilities and components of PEST. However to get the most out of PEST you should take the time to read this manual in its entirety. Parameter estimation is a "tricky business" and will not work unless you know what you are doing. If PEST does not appear to be able to calibrate your model or turn your model into a powerful data interpretation package, the chances are that you are misusing it. Thus, even though it may be heavy going, you should pay particular attention to Chapter 2 which provides details of the PEST algorithm. As was explained above, PEST may need to be "tuned" to your estimation or interpretation problem. All that it may take to achieve this is the adjustment of a single optimisation control variable, the log-transformation of a single parameter, or the setting of a single derivative increment. Unless you are aware of the possibilities available to you for modifying PEST's operation to suit your particular problem, you may never use it to its full potential.

Chapter 3 discusses the interface between PEST and your model, describing how to make PEST template and instruction files. Chapter 4 teaches you how to write a PEST control file and discusses the effects that different control settings have on PEST's performance. Chapter 5 tells you how to run PEST; it also discusses problems that may arise as PEST executes, and how best to overcome them. Chapter 6 discusses predictive analysis while Chapter 7 discusses regularisation. Both or these aspects of parameterisation functionality are unique to PEST (at the time of writing) and help to make PEST so universally useful in calibrating models that simulate real-world systems.

Chapter 8 discusses an advanced aspect of PEST's performance, viz. its ability to use derivatives calculated by the model instead of calculating them itself through the process of finite parameter differences. In most cases of PEST usage the modeller need not be too familiar with the contents of this chapter, for it is only in special circumstances that PEST is able to take advantage of the fact that the model is able to supply it with "externally-calculated derivatives".

Parallel PEST is described in Chapter 9, while Chapter 10 details the PEST utilities, TEMPCHEK, INSCHEK, PESTCHEK, PESTGEN, PARREP, JACWRIT and PAR2PAR.

Chapter 11 discusses the sensitivity analyser SENSAN, together with its utility program SENSCHEK. Chapter 12 presents an example of the use of PEST in solving a practical data-interpretation problem. Chapter 13 answers some frequently asked questions.

2. The PEST Algorithm

This chapter discusses the mathematical foundations of the PEST nonlinear parameter estimation algorithm and the means by which this theory has been implemented in the construction of the powerful parameter optimiser which is PEST. However the discussion is brief and no proofs are presented. The reader is referred to the limited bibliography at the end of the chapter for a number of books which treat the subject in much greater detail.

2.1 The Mathematics of PEST

2.1.1 Parameter Estimation for Linear Models

Let us assume that a natural or man-made system can be described by the linear equation

$$\mathbf{Xb} = \mathbf{c} \tag{2.1}$$

In equation 2.1 **X** is a $m \times n$ matrix, ie. it is a matrix with m rows and n columns. The elements of **X** are constant and hence independent of the elements of **b**, a vector of order n which, we assume, holds the system parameters. **c** is a vector of order m containing numbers which describe the system's response to a set of excitations embodied in the matrix **X**, and for which we can obtain corresponding field or laboratory measurements by which to infer the system parameters comprising **b**. (Note that for many problems to which PEST is amenable, the system parameters may be contained in **X** and the excitations may comprise the elements of **b**. Nevertheless, in the discussion which follows, it will be assumed for the sake of simplicity that **b** holds the system parameters.)

The word "observations" will be used to describe the elements of the vector **c** even though **c** is, in fact, generated by the model. This is because most models generate a wealth of data for which we may have only a handful of corresponding field measurements on which to base our estimates of the system properties. Hence, as we include in the vector **c** only those model outcomes for which there are complementary laboratory or field measurements, it is appropriate to distinguish them from the remainder of the model outcomes by referring to them as the "model-generated observations". The complementary set of field or laboratory data is referred to as "measurements" or as "experimental observations" in the following discussion.

Let it be assumed that the elements of **X** are all known. For most models these elements will include the effects of such things as the system dimensions, physical, chemical or other constants which are considered immutable, independent variables such as time and distance etc. For example, equation 2.1 may represent the response of the system at different times, where the response at time p is calculated using the equation

$$x_{p1}\, b_1 + x_{p2}\, b_2 + ... + x_{pn}\, b_n = c_p \tag{2.2}$$

where x_{pi} is the element of **X** found at its p'th row and i'th column. As **X** has m rows, there are m such equations, one for each of m different times. Hence for any p, at least one of the x_{pi} depends on time.

Suppose that m is greater than n, ie. we are capable of observing the system response (and hence providing elements for the vector **c**) at more times than there are parameters in the vector **b**. Common sense tells us that we should be able to use the elements of **c** to infer the elements of **b**.

Unfortunately we cannot do this by recasting equation 2.1 as another matrix equation with **b** on the right-hand side, as **X** is not a square matrix and hence not directly invertible. But you may ask "Have we not made a rod for our own back by measuring the system response at more times than there are parameter values, ie. elements of **b**?" If **b** was of the same order as **c, X** would indeed be a square matrix and may well be invertible. If so, it is true that an equation could be formulated which solves for the elements of **b** in terms of those of **c**. However, what if we then made just one more measurement of the system at a time not already represented in the $n \times n$ matrix **X**? We would now have $n + 1$ values of **c**; which n of these would we use in solving for **b**? And what would we do if we obtained (as we probably would) slightly different estimates for the components of **b** depending on which n of the $n + 1$ values of **c** we used in solving for **b**? The problem becomes even more acute if the information redundancy is greater than one.

Actually, as intuition should readily inform us, redundancy of information is a bonus rather than a problem, for it allows us to determine not just the elements of **b**, but some other numbers which describe how well we can trust the elements of **b**. This "trustworthiness" is based on the consistency with which the m experimental measurements satisfy the m equations expressed by equation 2.1 when the n optimal parameter values are substituted for the elements of **b**.

We define this optimal parameter set as that for which the sum of squared deviations between model-generated observations and experimental observations is reduced to a minimum; the smaller is this number (referred to as the "objective function") the greater is the consistency between model and observations and the greater is our confidence that the parameter set determined on the basis of these observations is the correct one. Expressing this mathematically, we wish to minimise Φ, where Φ is defined by the equation

$$\Phi = (\mathbf{c} - \mathbf{Xb})^t(\mathbf{c} - \mathbf{Xb}), \qquad (2.3)$$

and **c** now contains the set of laboratory or field measurements; the "t" superscript indicates the matrix transpose operation. It can be shown that the vector **b** which minimises Φ of equation 2.3 is given by

$$\mathbf{b} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{c}. \qquad (2.4)$$

Provided that the number of observations m equals or exceeds the number of parameters n, the matrix equation 2.4 provides a unique solution to the parameter estimation problem. Furthermore, as the matrix $(\mathbf{X}^t\mathbf{X})$ is positive definite under these conditions, the solution is relatively easy to obtain numerically.

The vector **b** expressed by equation 2.4 differs from **b** of equation 2.1 (the equation which defines the system) in that the former is actually an estimate of the latter because **c** now contains measured data. In fact, **b** of equation 2.4 is the "best linear unbiased" estimator of the set of true system parameters appearing in equation 2.1. As an estimator, it is one

particular realisation of the *n*-dimensional random vector **b** calculated, through equation 2.4, from the *m*-dimensional random vector **c** of experimental observations, of which the actual observations are but one particular realisation**.** If σ^2 represents the variance of each of the elements of **c** (the elements of **c** being assumed to be independent of each other) then σ^2 can be calculated as

$$\sigma^2 = \Phi/(m - n) \tag{2.5}$$

where (m - n), the difference between the number of observations and the number of parameters to be estimated, represents the number of "degrees of freedom" of the parameter estimation problem. Equation 2.5 shows that σ^2 is directly proportional to the objective function and thus varies inversely with the goodness of fit between experimental data and the model-generated observations calculated on the basis of the optimal parameter set. It can further be shown that C(**b**), the covariance matrix of **b** is given by

$$C(\mathbf{b}) = \sigma^2(\mathbf{X}^t\mathbf{X})^{-1} \tag{2.6}$$

Notice that, even though the elements of **c** are assumed to be independent (so that the covariance matrix of **c** contains only diagonal elements, all equal to σ^2 in the present case), C(**b**) is not necessarily a diagonal matrix. In fact, in many parameter estimation problems parameters are strongly correlated, the estimation process being better able to estimate one or a number of linear combinations of the parameters than the individual parameters themselves. In such cases some parameter variances (parameter variances constitute the diagonal elements of C(**b**)) may be large even though the objective function Φ is reasonably low. If parameter correlation is extreme, the matrix ($\mathbf{X}^t\mathbf{X}$) of equation 2.6 may become singular and parameter estimation becomes impossible.

There are two matrices, both of which are derived from the parameter covariance matrix C(**b**), which better demonstrate parameter correlation than C(**b**) itself. The first is the correlation coefficient matrix whose elements ρ_{ij} are calculated as

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\,\sigma_{jj}}} \tag{2.7}$$

where σ_{ij} represents the element at the *i*'th row and *j*'th column of C(**b**). The diagonal elements of the correlation coefficient matrix are always 1; off-diagonal elements range between -1 and 1. The closer are these off-diagonal elements to 1 or -1, the more highly are the respective parameters correlated.

The second useful matrix is comprised of columns containing the normalised eigenvectors of the covariance matrix C(**b**). If each eigenvector is dominated by one element, individual parameter values may be well resolved by the estimation process. However if predominance within each eigenvector is shared between a number of elements (especially for those eigenvectors whose eigenvalues are largest), the corresponding parameters are highly correlated. See Section 5.2.11 for further details.

2.1.2 Observation Weights

The discussion so far presupposes that all observations carry equal weight in the parameter estimation process. However this will not always be the case as some measurements may be more prone to experimental error than others.

Another problem arises where observations are of more than one type. For example equation 2.1 may represent a plant growth model; you may have a set of biomass and soil moisture content measurements which you would like to use to estimate some parameters for the model. However the units for these two quantities are different (kg/ha and dimensionless respectively); hence the numbers used to represent them may be of vastly different magnitude. Under these circumstances the quantity represented by the larger numbers will take undue precedence in the estimation process if the objective function is defined by equation 2.3; this will be especially unfortunate if the quantity represented by the smaller numbers is, in fact, measured with greater reliability than that represented by the larger numbers.

This problem can be overcome if a weight is supplied with each observation; the larger the weight pertaining to a particular observation the greater the contribution that the observation makes to the objective function. If the observation weights are housed in an m-dimensional, square, diagonal matrix **Q** whose i'th diagonal element q_{ii} is the square of the weight w_i attached to the i'th observation, equation 2.3 defining the objective function is modified as follows:

$$\Phi = (\mathbf{c} - \mathbf{Xb})^t \mathbf{Q}(\mathbf{c} - \mathbf{Xb}) \qquad (2.8a)$$

or, to put it another way,

$$\Phi = \sum_{i=1}^{m} (w_i r_i)^2 \qquad (2.8b)$$

where r_i (the i'th residual) expresses the difference between the model outcome and the actual field or laboratory measurement for the i'th observation. If observation weights are correctly assigned, it can be shown that equation 2.8a is equivalent to

$$\Phi = (\mathbf{c} - \mathbf{Xb})^t \mathbf{P}^{-1}(\mathbf{c} - \mathbf{Xb}) \qquad (2.9)$$

where

$$\mathbf{P} \ (= \mathbf{Q}^{-1}) = C(\mathbf{c})/\sigma^2 \qquad (2.10)$$

$C(\mathbf{c})$ represents the covariance matrix of the m-dimensional observation random vector **c** of which our measurement vector is a particular realisation. Because **Q** is a diagonal matrix, so too is **P**, its elements being the reciprocals of the corresponding elements of **Q**. The assumption of independence of the observations is maintained through insisting that **Q** (and hence **P**) have diagonal elements only, the elements of **Q** being the squares of the observation weights. These weights can now be seen as being inversely proportional to the standard deviations of the field or laboratory measurements to which they pertain. (Note that the weights as defined by equation 2.8 are actually the square roots of the weights as defined by

some other authors. However they are defined as such herein because it has been found that users, when assigning weights to observations, find it easier to think in terms of standard deviations than variances, especially when dealing with two or three different observation types of vastly different magnitude.)

The quantity σ^2 is known as the reference variance; if all observation weights are unity it represents the variance of each experimental measurement. If the weights are not all unity the measurement covariance matrix is determined from equation 2.10 with σ^2 given by equation 2.5 and Φ given by equation 2.8.

With the inclusion of observation weights, equation 2.4 by which the system parameter vector is estimated becomes

$$\mathbf{b} = (\mathbf{X}^t\mathbf{Q}\mathbf{X})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{c} \qquad (2.11)$$

while equation 2.6 for the parameter covariance matrix becomes

$$C(\mathbf{b}) = \sigma^2(\mathbf{X}^t\mathbf{Q}\mathbf{X})^{-1} \qquad (2.12)$$

2.1.3 The Use of Prior Information in the Parameter Estimation Process

It often happens that we have some information concerning the parameters that we wish to optimise, and that we obtained this information independently of the current experiment. This information may be in the form of other, unrelated, estimates of some or all of the parameters, or of relationships between parameters expressed in the form of equation 2.2. It is often useful to include this information in the parameter estimation process both for the philosophical reason that it is a shame to withhold it, and because this information may lend stability to the process. The latter may be the case where parameters, as determined solely from the current experiment, are highly correlated. This can lead to nonunique parameter estimates because certain pairs or groups of parameters, if varied in concert in a certain linear combination, may effect very little change in the objective function. In some cases this nonuniqueness can even lead to numerical instability and failure of the estimation process. However if something is known about at least one of the members of such a troublesome parameter group, this information, if included in the estimation process, may remove the nonuniqueness and provide stability.

Parameter estimates will also be nonunique if there are less observations then there are parameters; equation 2.11 is not solvable under these conditions as the matrix $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ is singular. (Note that PEST will, nevertheless, calculate parameter estimates for reasons discussed later in this chapter.) However the inclusion of prior information, being mathematically equivalent to taking extra measurements, may alter the numerical predominance of parameters over observations and thus provide the system with the ability to supply a unique set of parameter estimates.

Prior information is included in the estimation algorithm by simply adding rows containing this information to the matrix equation 2.1. This information must be of a suitable type to be included in equation 2.1; both simple equality, and linear relationships of the type described by equation 2.2 are acceptable. A weight must be included with each article of prior information, this weight being inversely proportional to the standard deviation of the right

hand side of the prior information equation, the constant of proportionality being the same as used for the observations comprising the other elements of the vector **c** of equation 2.1. In practice, the user simply assigns the weights in accordance with the extent to which he/she wishes each article of prior information to influence the parameter estimation process.

It is sometimes helpful to view the inclusion of prior parameter information in the estimation process as the introduction of one or more "penalty functions". The aim of the estimation process is to lower the objective function defined by equation 2.8 to its minimum possible value; this is done by adjusting parameter values until a set is found for which the objective function can be lowered no further. If there is no prior information, the objective function is defined solely in terms of the discrepancies between model outcomes and laboratory or field measurements. However with the inclusion of prior information, minimising the discrepancy between model calculations and experimental measurements is no longer the sole aim of the parameter estimation process. To the extent that any article of prior information is not satisfied, there is introduced into the objective function a "penalty" equal to the squared discrepancy between what the right hand side of the prior information equation should be, and what it actually is according to the current set of parameter values. This discrepancy is multiplied by the squared weight pertaining to that article of prior information prior to inclusion in the objective function.

2.1.4 Nonlinear Parameter Estimation

Most models are nonlinear, ie. the relationships between parameters and observations are not of the type expressed by equations 2.1 and 2.2. Nonlinear models must be "linearised" in order that the theory presented so far can be used in the estimation of their parameters.

Let the relationships between parameters and model-generated observations for a particular model be represented by the function M which maps n-dimensional parameter space into m-dimensional observation space. For reasons which will become apparent in a moment, we require that this function be continuously differentiable with respect to all model parameters for which estimates are sought. Suppose that for the set of parameters comprising the vector $\mathbf{b}_0$ the corresponding set of model-calculated observations (generated using M) is $\mathbf{c}_0$, ie.

$$\mathbf{c_0} = M(\mathbf{b_0}). \tag{2.13}$$

Now to generate a set of observations **c** corresponding to a parameter vector **b** that differs only slightly from $\mathbf{b}_0$, Taylor's theorem tells us that the following relationship is approximately correct, the approximation improving with proximity of **b** to $\mathbf{b}_0$:

$$\mathbf{c} = \mathbf{c}_0 + \mathbf{J}(\mathbf{b} - \mathbf{b}_0) \tag{2.14}$$

where **J** is the Jacobian matrix of M, ie. the matrix comprised of m rows (one for each observation), the n elements of each row being the derivatives of one particular observation with respect to each of the n parameters. To put it another way, $\mathbf{J}_{ij}$ is the derivative of the i'th observation with respect to the j'th parameter. Equation 2.14 is a linearisation of equation 2.13.

We now specify that we would like to derive a set of model parameters for which the model-generated observations are as close as possible to our set of experimental observations in the

least squares sense, ie. we wish to determine a parameter set for which the objective function Φ defined by

$$\Phi = (\mathbf{c} - \mathbf{c}_0 - \mathbf{J}(\mathbf{b} - \mathbf{b}_0))^t\mathbf{Q}\,(\mathbf{c} - \mathbf{c}_0 - \mathbf{J}(\mathbf{b} - \mathbf{b}_0)) \qquad (2.15)$$

is a minimum, where **c** in equation 2.15 now represents our experimental observation vector. Comparing equation 2.15 with equation 2.8, it is apparent that the two are equivalent if **c** from equation 2.8a is replace by ($\mathbf{c} - \mathbf{c}_0$) of equation 2.15 and **b** from equation 2.8a is replaced by ($\mathbf{b} - \mathbf{b}_0$) from equation 2.15. Thus we can use the theory that has been presented so far for linear parameter estimation to calculate the parameter upgrade vector ($\mathbf{b} - \mathbf{b}_0$) on the basis of the vector ($\mathbf{c} - \mathbf{c}_0$) which defines the discrepancy between the model-calculated observations $\mathbf{c}_0$ and their experimental counterparts **c**. Denoting **u** as the parameter upgrade vector, equation 2.11 becomes

$$\mathbf{u} = (\mathbf{J}^t\mathbf{Q}\mathbf{J})^{-1}\mathbf{J}^t\mathbf{Q}(\mathbf{c} - \mathbf{c}_0) \qquad (2.16)$$

and equation 2.12 for the parameter covariance matrix becomes

$$C(\mathbf{b}) = \sigma^2(\mathbf{J}^t\mathbf{Q}\mathbf{J})^{-1} \qquad (2.17)$$

The linear equations represented by the matrix equation 2.16 are often referred to as the "normal equations". The matrix ($\mathbf{J}^t\mathbf{Q}\mathbf{J}$) is often referred to as the "normal matrix".

Because equation 2.14 is only approximately correct, so too is equation 2.16; in other words, the vector **b** defined by adding the parameter upgrade vector **u** of equation 2.16 to the current parameter values $\mathbf{b}_0$ is not guaranteed to be that for which the objective function is at its minimum. Hence the new set of parameters contained in **b** must then be used as a starting point in determining a further parameter upgrade vector, and so on until, hopefully, we arrive at the global Φ minimum. This process requires that an initial set of parameters $\mathbf{b}_0$ be supplied to start off the optimisation process. The process of iterative convergence towards the objective function minimum is represented diagrammatically for a two-parameter problem in Figure 2.1.

It is an unfortunate fact in working with nonlinear problems, that a global minimum in the objective function may be difficult to find. For some models the task is made no easier by the fact that the objective function may even possess local minima, distinct from the global minimum. Hence it is always a good idea to supply an initial parameter set $\mathbf{b}_0$ that you consider to be a good approximation to the true parameter set. A suitable choice for the initial parameter set can also reduce the number of iterations necessary to minimise the objective function; for large models this can mean considerable savings in computer time. Also, the inclusion of prior information into the objective function can change its structure in parameter space, often making the global minimum easier to find (depending on what weights are applied to the articles of prior information). Once again, this enhances optimisation stability and may reduce the number of iterations required to determine the optimal parameter set.

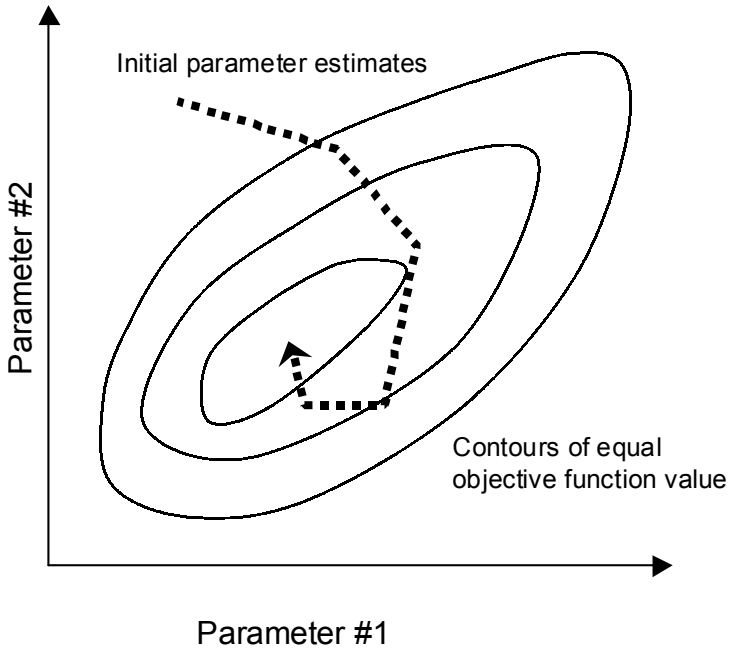


Figure 2.1 Iterative improvement of initial parameter values toward the global objective function minimum.

2.1.5 The Marquardt Parameter

Equation 2.16 forms the basis of nonlinear weighted least squares parameter estimation. It can be rewritten as

$$\mathbf{u} = (\mathbf{J}^t\mathbf{Q}\mathbf{J})^{-1}\mathbf{J}^t\mathbf{Q}\mathbf{r} \qquad (2.18)$$

where **u** is the parameter upgrade vector and **r** is the vector of residuals for the current parameter set.

Let the gradient of the objective function Φ in parameter space be denoted by the vector **g**. The *i*'th element of **g** is thus defined as

$$g_i = \frac{\partial \Phi}{\partial b_i} \qquad (2.19)$$

ie. by the partial derivative of the objective function with respect to the *i*'th parameter. The parameter upgrade vector cannot be at an angle of greater than 90 degrees to the negative of the gradient vector. If the angle between **u** and **-g** is greater than 90 degrees, **u** would have a component along the positive direction of the gradient vector and movement along **u** would thus cause the objective function to rise, which is the opposite of what we want. However, in spite of the fact that **-g** defines the direction of steepest descent of Φ, it can be shown that **u** is normally a far better parameter upgrade direction than **-g**, especially in situations where parameters are highly correlated. In such situations, iteratively following the direction of steepest descent leads to the phenomenon of "hemstitching" where the parameter set jumps from side to side of a valley in Φ as parameters are upgraded on successive iterations;

convergence toward the global Φ minimum is then extremely slow. See Figure 2.2.

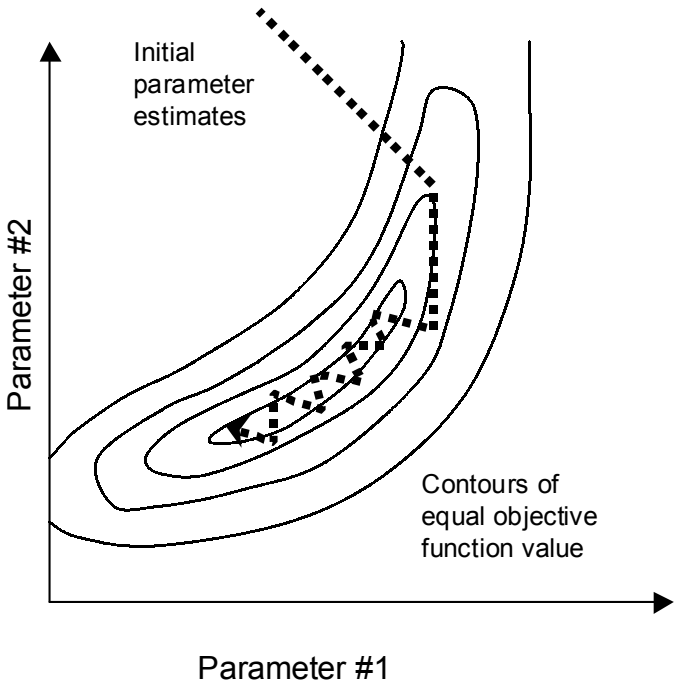


Figure 2.2 The phenomenon of "hemstitching".

Nevertheless, most parameter estimation problems benefit from adjusting **u** such that it is a little closer to the direction of **-g** in the initial stages of the estimation process. Mathematically, this can be achieved by including in equation 2.18 the so-called "Marquardt parameter", named after Marquardt (1963), though the use of this parameter was, in fact, pioneered by Levenberg (1944). Equation 2.18 becomes

$$\mathbf{u} = (\mathbf{J}^t\mathbf{Q}\mathbf{J} + \alpha\mathbf{I})^{-1}\mathbf{J}^t\mathbf{Q}\mathbf{r} \qquad (2.20)$$

where α is the Marquardt parameter and **I** is the $n \times n$ identity matrix.

It can be shown that the gradient vector **g** can be expressed as

$$\mathbf{g} = -2\mathbf{J}^t\mathbf{Q}\mathbf{r} \qquad (2.21)$$

It follows from equations 2.20 and 2.21 that when α is very high the direction of **u** approaches that of the negative of the gradient vector; when α is zero, equation 2.20 is equivalent to equation 2.18. Thus for the initial optimisation iterations it is often beneficial for α to assume a relatively high value, decreasing as the estimation process progresses and the optimum value of Φ is approached. The manner in which PEST decides on a suitable value for α for each iteration is discussed in Section 2.1.7.

2.1.6 Scaling

For many problems, especially those involving different types of observations and parameters whose magnitudes may differ greatly, the elements of **J** can be vastly different in magnitude.

This can lead to roundoff errors as the upgrade vector is calculated through equation 2.20. Fortunately, this can be circumvented to some extent through the use of a scaling matrix **S**. Let **S** be a square, $n \times n$ matrix with diagonal elements only, the i'th diagonal element of **S** being given by

$$S_{ii} = (\mathbf{J}^t\mathbf{Q}\mathbf{J})_{ii}^{-1/2} \qquad (2.22)$$

Introducing **S** into equation 2.20 the following equation can be obtained for $\mathbf{S}^{-1}\mathbf{u}$:

$$\mathbf{S}^{-1}\mathbf{u} = ((\mathbf{J}\mathbf{S})^t\mathbf{Q}\mathbf{J}\mathbf{S} + \alpha\mathbf{S}^t\mathbf{S})^{-1}(\mathbf{J}\mathbf{S})^t\mathbf{Q}\mathbf{r} \qquad (2.23)$$

It can be shown that although equation 2.23 is mathematically equivalent to equation 2.20 it is numerically far superior.

If α is zero, the matrix $(\mathbf{J}\mathbf{S})^t\mathbf{Q}\mathbf{J}\mathbf{S} + \alpha\mathbf{S}^t\mathbf{S}$ has all its diagonal elements equal to unity. For a non-zero α the diagonal elements of $(\mathbf{J}\mathbf{S})^t\mathbf{Q}\mathbf{J}\mathbf{S} + \alpha\mathbf{S}^t\mathbf{S}$ will be greater than unity, though in general they will not be equal. Let the largest element of $\alpha\mathbf{S}^t\mathbf{S}$ be denoted as λ, referred to henceforth as the "Marquardt lambda". Then the largest diagonal element of the scaled normal matrix $(\mathbf{J}\mathbf{S})^t\mathbf{Q}\mathbf{J}\mathbf{S} + \alpha\mathbf{S}^t\mathbf{S}$ of equation 2.23 will be $1 + \lambda$.

2.1.7 Determining the Marquardt Lambda

PEST requires that the user supply an initial value for λ. During the first optimisation iteration PEST solves equation 2.23 for the parameter upgrade vector **u** using that user-supplied λ. It then upgrades the parameters, substitutes them into the model, and evaluates the resulting objective function. PEST then tries another λ, lower by a user-supplied factor than the initial λ. If Φ is lowered, λ is lowered yet again. However if Φ was raised by reducing λ below the initial λ, then λ is raised above the initial lambda by the same user-supplied factor, a new set of parameters is obtained through solution of equation 2.23, and a new Φ is calculated. If Φ was lowered, λ is raised again. PEST uses a number of different criteria to determine when to stop testing new λ's and proceed to the next optimisation iteration; see Section 4.2.2. Normally between one and four λ's need to be tested in this manner per optimisation iteration.

At the next iteration PEST repeats the procedure, using as its starting λ either the λ from the previous iteration that provided the lowest Φ (if λ needed to be raised from its initial value to achieve this Φ) or the previous iteration's best λ reduced by the user-supplied factor. In most cases this process results in an overall lowering of λ as the estimation process progresses.

Testing the effects of a few different λ's in this manner requires that PEST undertake a few extra model runs per optimisation iteration; however this process makes PEST very "robust". If the optimisation procedure appears to be "bogged", the adjustments made to λ in this fashion often result in the determination of a parameter upgrade vector that gets the process moving again.

2.1.8 Optimum Length of the Parameter Upgrade Vector

Inclusion of the Marquardt parameter in equation 2.23 has the desired effect of rotating the parameter upgrade vector **u** towards the negative of the gradient vector. However while the

direction of **u** may now be favourable, its magnitude may not be optimum.

Under the linearity assumption used in deriving all equations presented so far, it can be shown that the optimal parameter adjustment vector is given by $\beta\mathbf{u}$, where **u** is determined using equation 2.23 and β is calculated as

$$\beta = \frac{\sum_{i=1}^{m} (c_i - c_{0i}) w_i^2 \gamma_i}{\sum_{i=1}^{m} (w_i \gamma_i)^2} \qquad (2.24)$$

where, once again, the vector **c** represents the experimental observations, $\mathbf{c}_0$ represents their current model-calculated counterparts, w_i is the weight pertaining to observation i, and γ_i is given by

$$\gamma_i = \sum_{j=1}^{n} u_j \frac{\partial c_{0i}}{\partial b_j} \qquad (2.25a)$$

ie.

$$\gamma = \mathbf{Ju} \qquad (2.25b)$$

where **J** represents the Jacobian matrix once again. If $\mathbf{b}_0$ holds the current parameter set the new, upgraded set is calculated using the equation

$$\mathbf{b} = \mathbf{b}_0 + \beta\mathbf{u} \qquad (2.26)$$

2.1.9 Predictive Analysis

Let **X** represent the action of a linear model under calibration conditions. Let **b** represent the parameter vector for this model, while **c** is a vector of field or laboratory observations for which there are model-generated counterparts. As is explained in Section 2.1.2, when calibrating a model, it is PEST's task to minimise an objective function defined as

$$\Phi = (\mathbf{c} - \mathbf{Xb})^t \mathbf{Q} (\mathbf{c} - \mathbf{Xb})$$

where **Q** is the "cofactor matrix", a diagonal matrix whose elements are the squares of observation weights.

Let **Z** represent the same linear model under predictive conditions. Thus the action of the model when used in predictive mode can be represented by the equation

$$d = \mathbf{d} = \mathbf{Zb} \qquad (2.27)$$

where **d** is a 1×1 vector (ie. a scalar, d) representing a single model outcome (ie. prediction). Naturally, when run in predictive mode, the model operates on the same parameter vector as that on which it operates in calibration mode, ie. **b**.

The aim of predictive analysis is to maximise (minimise) *d* while "keeping the model calibrated". *d* will be maximised (minimised) when the objective function associated with **b** lies on the $\Phi_{min} + \delta$ contour. Φ_{min} is the lowest achievable value of the objective function, while δ is an acceptable increment to the objective function minimum such that the model can be considered calibrated as long as the objective function is less than $\Phi_{min} + \delta$; see the discussion in Chapter 6 for further details. Thus the predictive analysis problem can be formulated as follows:-

Find **b** such as to maximise (minimise)

$$\mathbf{Zb} \qquad (2.28a)$$

subject to

$$(\mathbf{c} - \mathbf{Xb})^t\mathbf{Q}(\mathbf{c} - \mathbf{Xb}) = \Phi_0 \qquad (2.28b)$$

where

$$\Phi_0 = \Phi_{min} + \delta \qquad (2.28c)$$

It can be shown that the solution to this problem is

$$\mathbf{b} = \left(\mathbf{X}^T\mathbf{Q}\mathbf{X}\right)^{-1}\left\{\mathbf{X}^t\mathbf{Q}\mathbf{c} - \frac{\mathbf{Z}}{2\lambda}\right\} \qquad (2.29a)$$

where λ is defined by the equation

$$\left(\frac{1}{2\lambda}\right)^2 = \frac{\Phi_0 - \mathbf{c}^t\mathbf{Q}\mathbf{c} + \mathbf{c}^t\mathbf{Q}\mathbf{X}\left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{c}}{\mathbf{Z}^t\left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\mathbf{Z}} \qquad (2.29b)$$

Where predictive analysis is carried out for a nonlinear model the same equations are used. However in this case **X** is replaced by the model Jacobian matrix **J**, and a parameter upgrade vector is calculated instead of a solution vector. The solution process is then an iterative one in which the true solution is approached by repeated calculation of an upgrade vector based on repeated linearisation of the problem through determination of the Jacobian matrix. For further details see Cooley and Vecchia (1987) and Vecchia and Cooley (1987).

Use of the Marquardt lambda in solving the nonlinear parameter estimation problem is discussed above. Its use in solving the nonlinear predictive analysis problem is very similar. As in the parameter estimation problem, PEST continually adjusts the Marquardt lambda through the solution process such that its value is optimal at all stages of this process.

As a further numerical measure to solve the predictive analysis problem for a nonlinear model, PEST undertakes a line search in the direction of the parameter upgrade vector to determine the point at which this vector crosses the Φ_0 contour. See Chapter 6 for further details.

2.1.10 Regularisation

The theory which underpins PEST's regularisation functionality bears some resemblance to that which underpins its predictive analysis functionality. As is explained in Section 2.1.2, when calibrating a model it is PEST's task to minimise an objective function defined as

$$\Phi_m = (\mathbf{c} - \mathbf{Xb})^t \mathbf{Q}_m (\mathbf{c} - \mathbf{Xb}) \qquad (2.30a)$$

The "m" subscript introduced to the left side of equation 2.30, which is otherwise equivalent to equation 2.8, denotes the fact that the objective function is comprised of the sum of weighted squared differences between model outputs and field measurements (the "m" stands for "measurement"). The vector **c** is comprised of field measurements, while the vector **b** is comprised of the parameters which must be estimated. As has already been mentioned, $\mathbf{Q}_m$ is a diagonal matrix whose elements are the squares of measurement weights.

We wish to impose the requirement that a "regularisation objective function" Φ_r be also minimised by the parameter set which is estimated by PEST. Φ_r is defined by the equation

$$\Phi_r = (\mathbf{d} - \mathbf{Zb})^t \mathbf{Q}_r (\mathbf{d} - \mathbf{Zb}) \qquad (2.30b)$$

where $\mathbf{Q}_r$ is a diagonal matrix comprised of the squares of weights assigned to the various "regularisation observations" which collectively comprise the vector **d**. The relationships by which the model-generated counterparts to these "observations" are calculated from the parameter values (constituting the vector **b**) are encapsulated in the matrix **Z**. (Note that if these relationships are not linear, a linear approximation to them can be calculated by taking derivatives with respect to parameters in the same way that the Jacobian matrix is calculated as a linear approximation to the model function.) As an example, each row of the matrix **Z** may be comprised of 0's, except for two elements which are 1 and -1; the 1 and -1 elements occur at those of its columns which pertain to two parameters whose difference is taken. The "observed value" of this difference (ie. the element of the corresponding row of the vector **d**) might be zero, indicating that PEST should minimise this parameter difference insofar as this is possible while still allowing the model to fit the field data. If there are as many rows in the matrix **Z** as there are parameters in the model domain, and if each such row represents a single parameter difference, and if the "observed" difference is zero in each case, then imposition of the regularisation criterion is the imposition of a "maximum homogeneity" condition. There may, in fact, be more differences represented in the matrix **Z** than there are parameters in the model domain. This may occur if, for example, differences are formed in both the horizontal and vertical directions (or row and column directions) of a particular model domain to ensure maximum parameter uniformity in both of these directions. Similar matrices can be formed in order to impose a minimum curvature constraint or to implement any other type of regularisation criteria that the user may wish to impose (see Chapter 7 for more details).

When working in "regularisation mode", PEST's task is to minimise Φ_r while ensuring that Φ_m is "suitably low". Such a "suitably low" value will normally be slightly above the minimum value for Φ_m that could have been achieved if regularisation conditions had not been imposed. It is the user's responsibility to select this value; it will be denoted as Φ_m^l (ie. the "limiting measurement objective function"). It is this ability which PEST gives the user to indicate in advance of the parameter estimation process the extent of model-to-measurement

misfit that is tolerable in order to attempt to satisfy imposed regularisation constraints on parameter values that makes the regularisation methodology, as implemented by PEST, so powerful. Thus the regularisation process must minimise Φ_r while enforcing the condition that

$$\Phi_m \leq \Phi_m^l$$

or, in practice, that

$$\Phi_m = \Phi_m^l \tag{2.31}$$

because a decrease in Φ_r will nearly always require an increase in Φ_m where parameter values are close to optimum.

The constrained minimisation problem described above can be formulated as an unconstrained minimisation problem through the use of a Lagrange multiplier γ. Thus, with regularisation constraints imposed, the parameterisation problem consists in determining the elements of the vector **b** (ie. the values of the adjustable parameters) which minimise the "total" objective function Φ_t defined by the equation:

$$\Phi_t = \Phi_r + \gamma\Phi_m \tag{2.32}$$

while simultaneously finding the value for γ which causes equation 2.31 to be satisfied.

As was mentioned above, this problem is somewhat similar to the predictive analysis problem discussed in the previous section in that one function is minimised (in this case the regularisation objective function) while the objective function based on field or laboratory measurements is held at some upper limit selected by the user below which the model is deemed to be calibrated.

An inspection of equation 2.32 reveals that the regularisation problem can be viewed from a slightly different angle. It can be formulated as a traditional nonlinear parameter estimation problem which attempts to estimate the parameter set **b** that provides the best fit between a set of observations and corresponding model outputs, with the observations consisting of both "measurement observations" and "regularisation observations". The Lagrange multiplier γ can then be considered as a factor by which to multiply the weights pertaining to the measurement observations in order to ensure that equation 2.31 is satisfied when the total objective function Φ_t is minimised. Alternatively, if equation 2.32 is divided by γ, the parameter estimation problem can also be seen as the problem of minimising Φ defined as

$$\Phi = \mu\Phi_t = \mu\Phi_r + \Phi_m \tag{2.33}$$

where μ is the reciprocal of γ. With the objective function defined in this way, the reciprocal of the Lagrange multiplier can be seen to be equivalent to a "regularisation weight factor", ie. the factor by which all "regularisation observations" are multiplied in formulation of the overall objective function, now defined as Φ. Once again, the value of μ must be such that equation 2.31 is satisfied.

When undertaking problem regularisation, PEST minimises the objective function Φ defined by equation 2.33. During each optimisation iteration it calculates the regularisation weight

factor μ that results in equation 2.31 being satisfied. It does this using an iterative procedure based on linearised model and regularisation conditions. Once μ has been determined, all regularisation weights (ie. weights assigned by the user to the regularisation observations) are multiplied by this factor; a parameter upgrade vector is then calculated in the normal way. The problem is then linearised again (through calculation of a new Jacobian matrix) and the process is repeated.

2.1.11 Use of an Observation Covariance Matrix

In the theory presented in Section 2.1.2, the squares of user-supplied observation weights comprise the elements of the diagonal matrix **Q** (often referred to as the "cofactor matrix"). The inverse of **Q** is proportional to the covariance matrix of the observations, the constant of proportionality being the "reference variance". Because **Q** is a diagonal matrix, so too is the observation covariance matrix.

The use of observation weights in calculating the objective function is based on the premise that observations are independent, ie. that the "uncertainty" pertaining to any one observation bears no relationship to the "uncertainty" pertaining to any other observation. In practice, observation "uncertainty" in a calibration context is determined by the level of misfit between these observations and corresponding model outputs, ie. by the model-to-measurement residuals calculated at the end of the inversion process. If these residuals are expected to be uncorrelated, then observation uncertainties can be expressed in terms of individual observation weights. However if residuals are likely to show consistency over space and/or time for certain observation types, then it may not be appropriate to assume statistical independence for these observation types. 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.

The theory underpinning the use of observation weights presented in Section 2.1.2 is also applicable to the use of an observation covariance matrix in place of individual observation weights if the "cofactor matrix" **Q** is calculated as the inverse of a user-supplied observation covariance matrix. Note however that, as was mentioned above, a user-supplied observation covariance matrix can only be considered as *proportional* to the true observation covariance matrix; the latter can be determined after the inversion process is complete by multiplying the user-supplied covariance matrix by the reference variance determined through equation 2.5. At any stage of the optimisation process the objective function is computed using equation 2.8a; this is equivalent to equation 2.8b when **Q** is a diagonal matrix whose elements are the squares of observation weights.

Use of PEST's regularisation functionality does not preclude use of a covariance matrix to characterise either or both of measurement and regularisation observations. However when PEST is used in this mode the covariance matrix supplied for the regularisation observations must be separate from that supplied for the measurement observations. If a covariance matrix is provided for the regularisation observations, PEST will calculate a "weight factor" by which to multiply the "regularisation cofactor matrix" $\mathbf{Q}_r$ (calculated by PEST as the inverse of the user-supplied regularisation covariance matrix), in order to satisfy the goal of the regularisation process, ie. that the measurement component of the objective function be no

greater than the user-supplied threshold value of Φ_m^l while minimising the regularisation objective function Φ_r.

The remainder of this section describes the theory behind PEST's accommodation of the use of one or more observation covariance matrices in place of observation weights to characterise the uncertainty associated with groups of measurements and/or prior information equations. However, as the implementation of this theory within PEST takes place "behind the scenes", it is not essential to the use of PEST that this theory be fully understood.

Let **c** be a vector whose elements are stochastic variables, i.e. numbers with a random component. For the purposes of the present discussion, consider that the elements of **c** are the set of measurements to be used in the calibration process. Suppose that these measurements are statistically dependent on each other, and thus that the uncertainties associated with them must be represented by a covariance matrix rather than by a set of individual variances. Suppose that the covariance matrix associated with the elements of the vector **c** is the matrix **C**.

Because **C** is a covariance matrix, it must be positive definite (which means that it is also symmetric). Let **R** be a matrix whose columns are comprised of the normalised eigenvectors of **C.** It is easily shown that

$$\mathbf{R}^t = \mathbf{R}^{-1} \tag{2.34}$$

ie. that the transpose of **R** is also its inverse. Now let us introduce another stochastic vector **d**, calculable from the vector **c** using the relationship

$$\mathbf{d} = \mathbf{R}^t\mathbf{c} \tag{2.35}$$

It is easy to show that the covariance matrix of **d** (which will be named **D**) can be calculated from the covariance matrix of **c** (ie. **C**) using the relationship

$$\mathbf{D} = \mathbf{R}^t\mathbf{C}\mathbf{R} \tag{2.36}$$

and that **D** is a diagonal matrix whose elements are equal to the eigenvalues of **C**. This is a very important relationship, for it expresses the fact that through a simple rotational transformation of the original stochastic vector **c**, another stochastic vector **d** can be obtained whose elements are statistically independent. Hence if the "rotated" observation vector **d** is used as a basis for parameter estimation instead of the original observation vector **c**, weights can be used in the inversion equations instead of a covariance matrix.

Equation 2.11 describes how optimised parameter values (as encapsulated in the vector **b**) are calculated from measurements (ie. the vector **c**) for a linear model. (For the sake of simplicity, the present discussion is restricted to a linear model; the theory is easily extended to a nonlinear model using the methodology presented above.) The equation (repeated from equation 2.11) is

$$\mathbf{b} = (\mathbf{X}^t\mathbf{Q}\mathbf{X})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{c} \tag{2.37}$$

If the elements of the measurement vector **c** are not statistically independent, then the cofactor matrix **Q** of equation 2.37 has non-diagonal elements and, as is explained above, is

proportional to the inverse of the covariance matrix of **c**.

It is not too difficult to show that the vector of optimised parameter values **b** can also be calculated using the equation

$$\mathbf{b} = (\mathbf{Y}^t\mathbf{T}\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{T}\mathbf{d} \qquad (2.38)$$

where **d** is given by equation 2.35, and **Y** is calculated from the model matrix **X** using the equation

$$\mathbf{Y} = \mathbf{R}^t\mathbf{X} \qquad (2.39)$$

The matrix **T** in equation 2.38 is related to the inverse of the matrix **D** (the covariance matrix of **d**) by the same constant of proportionality that occurs in the relationship between the matrix **Q** of equation 2.37 and the inverse of the matrix **C** (the covariance matrix of **c)**. Equation 2.38 thus demonstrates that the vector **b** can be calculated from the rotated measurement vector **d** using exactly the same mathematics as that used to compute **b** from the non-rotated measurement vector **c**, provided that the model matrix **X** is also rotated, and that the cofactor matrix of **d** is used rather than the cofactor matrix of **c**. What is important however is that, for the reasons outlined above, **T** is a diagonal matrix whose elements are proportional to the inverse of the eigenvalues of **C** (and can thus be expressed as a set of weights). Equation 2.38 (with appropriate modifications for use in the context of a nonlinear model) is the one used by PEST to calculate optimised parameter values; however this is transparent to the user.

The relationship between the parameter covariance matrix **C**(**b**) and the observation cofactor matrix **Q** is expressed by equation 2.12, which is repeated below:-

$$\mathbf{C}(\mathbf{b}) = \sigma^2(\mathbf{X}^t\mathbf{Q}\mathbf{X})^{-1} \qquad (2.40)$$

Recall that σ^2 is the "reference variance". Equation 2.40 is applicable whether weights or a covariance matrix are used to specify observation uncertainties, ie. whether **Q** is a diagonal matrix or not. It is easily shown that the parameter covariance matrix can also be calculated from the rotated observation cofactor matrix **T** using the formula

$$\mathbf{C}(\mathbf{b}) = \sigma^2(\mathbf{Y}^t\mathbf{T}\mathbf{Y})^{-1} \qquad (2.41)$$

Recall that **T** is diagonal because rotation of the vector **c** to yield the vector **d** results in an uncorrelated set of stochastic variables.

It can also be shown that calculation of the objective function using equation 2.8a is equivalent to calculating it using the equation

$$\Phi = (\mathbf{d} - \mathbf{Y}\mathbf{b})^t\,\mathbf{T}(\mathbf{d} - \mathbf{Y}\mathbf{b}) \qquad (2.42)$$

Once again, calculations made using equation 2.42 are based on the use of a rotated observation dataset, (ie. the vector **d**) complemented by a rotated model matrix vector (ie. the matrix **Y**) and a rotated, diagonal, cofactor matrix **T**.

2.1.12 Goodness of Fit

When it is being used in parameter estimation mode, PEST aims to lower the objective function as far as it can be lowered. When used in predictive analysis mode, PEST aims to maximise or minimise a specified prediction while maintaining the model in a calibrated state, ie. while ensuring that the objective function rises no higher than a specified level. When working in regularisation mode PEST aims to maximise adherence to a certain "regularisation condition" (by minimising a regularisation objective function) while ensuring that the measurement objective function rises no higher than a specified level. In all of these cases, the extent to which model outputs are in agreement with their field-measured counterparts is apparent from the value of the objective function (or the "measurement objective function" when working in regularisation mode).

Another measure of goodness of fit is provided by the correlation coefficient as defined in Cooley and Naff (1990). Unlike the objective function, the correlation coefficient is independent of the number of observations involved in the parameter estimation process, and of the absolute levels of uncertainty associated with those observations. Hence use of this measure of goodness of fit allows the results of different parameter estimation exercises to be directly compared.

The correlation coefficient R is calculated as

$$R = \frac{\sum (w_i c_i - m)(w_i c_{oi} - m_o)}{\left[\sum (w_i c_i - m)(w_i c_i - m) \sum (w_i c_{oi} - m_o)(w_i c_{oi} - m_o)\right]^{1/2}} \qquad (2.43)$$

where:-

c_i is the i'th observation value,

c_{0i} is the model-generated counterpart to the i'th observation value,

m is the mean value of weighted observations,

m_o is the mean of weighted model-generated counterparts to observations, and

w_i is the weight associated with the i'th observation (or "rotated observation" if a covariance matrix is used to specify observation uncertainty instead of individual observation weights).

Generally R should be above 0.9 for the fit between model outputs and observations to be acceptable (Hill, 1998).

2.2 PEST's Implementation of the Method

So far, this chapter has discussed the theory behind PEST, viz. the method of weighted least squares and its application to nonlinear parameter estimation and predictive analysis. The remainder of this chapter discusses the ways in which the least squares method has been implemented in PEST to provide a general, robust, parameter estimation and predictive analysis package that is useable across a wide range of model types.

2.2.1 Parameter Transformation

PEST allows for the logarithmic transformation of some or all parameters. It often happens that if PEST (or any other parameter estimation program) is asked to estimate the log of a parameter, rather than the parameter itself, the process is much faster and more stable than it would otherwise be; however sometimes the opposite can occur.

PEST requires that each parameter be designated, in the PEST control file, as untransformed, log-transformed, fixed or tied; the latter two options will be discussed shortly. 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 pertaining to that parameter refer to the log of the parameter rather than to the parameter itself. However PEST parameter estimates and confidence intervals listed in the run record file refer to the actual parameter.

You should never ask PEST to logarithmically transform a parameter with a negative or zero initial value, or a parameter that may become negative or zero in the course of the estimation process. Hence a log-transformed parameter must be supplied with a positive lower bound (see below).

PEST is informed if a parameter is log-transformed through the parameter variable PARTRANS in the PEST control file; see Section 4.2.4. Note that more complex parameter transformations can be undertaken using the parameter preprocessor PAR2PAR; see Section 10.7.

2.2.2 Fixed and Tied Parameters

A parameter can be identified in a template file (see Chapter 3) yet take no part in the parameter estimation process. In this case it must be declared as "fixed" so that its value does not vary from that assigned to it as its initial estimate in the PEST control file.

PEST allows one or more parameters to be tied (ie. linked) to a "parent" parameter. PEST does not estimate a value for a tied parameter; rather it adjusts the parameter during the estimation process such that it maintains the same ratio with its parent parameter as that provided through the initial estimates of the respective 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.

PEST is informed whether a parameter is fixed or tied through the parameter variable PARTRANS in the PEST control file; see Section 4.2.4.

2.2.3 Upper and Lower Parameter Bounds

As well as supplying an initial estimate for each parameter, you are also required to supply parameter upper and lower bounds. These bounds define the maximum and minimum values which a parameter is allowed to assume during the optimisation process. They are provided through the parameter variables PARLBND and PARUBND in 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.

In some cases, where a large number of parameters are being estimated based on a large number of measurements, PEST may try to force a fit between model and measurements by adjusting some parameters to extremely large or extremely small values (especially if the measured values upon which the estimation process is based are not altogether consistent). Such extremely large or small values may, depending on the model, result in floating point errors or numerical convergence difficulties. Again, carefully chosen parameter bounds will circumvent this problem.

If a parameter upgrade vector **u** is determined which would cause one or more parameters to move beyond their bounds, PEST adjusts **u** such that this does not occur, placing such parameters at their upper or lower bounds. On later iterations, special treatment is then provided for parameters which are at their allowed limits. If the components of both the upgrade vector and the negative of the gradient vector pertaining to a parameter at its upper or lower limit are such as to take the parameter out of bounds, then the parameter is temporarily frozen, and the parameter estimation problem reformulated with that parameter fixed at its limit; hence the new upgrade vector will not result in any adjustment to that parameter. If, after reformulation of the problem in this manner, there are parameters at their limits for which the parameter upgrade vector still points outward, the negative of the gradient vector pointing inward, then these parameters, too, are temporarily frozen. This process continues until a parameter upgrade vector is calculated which either moves parameters from their bounds back into the allowed parameter domain, or leaves them fixed.

The strength of this strategy is that it allows PEST to search along the boundaries of the parameter domain for the smallest Φ to which it has access when the global minimum of Φ lies outside of the parameter domain, beyond PEST's reach.

At the beginning of each new optimisation iteration all temporarily-frozen parameters are freed to allow them to move back inside the allowed parameter domain if solution of equation 2.23 deems this necessary. The stepwise, temporary freezing of parameters is then repeated as described above.

2.2.4 Scale and Offset

For every parameter you must supply a scale and offset (variables SCALE and OFFSET in the PEST control file). Before writing a parameter value to a model input file, PEST multiplies this 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 redefine 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 optimise than elevation. In particular it may make more sense to express a derivative increment (see Section 2.3) as a fraction of thickness than as a fraction of elevation, to which an arbitrary datum has been added. Also, the optimisation process may be better behaved if the thickness parameter is log-transformed; again it would be surprising if the log-transformation of elevation improved optimisation performance due to the arbitrary datum with respect to which an elevation must be expressed. PEST can thus optimise thickness, converting this thickness to elevation every time it writes a model input file by adding the reference elevation stored as the parameter offset.

The scale variable is equally useful. A model parameter may be such that it can only take on 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 optimise this new parameter, log-transforming it if necessary to enhance optimisation efficiency. Just before it writes the parameter to a model input file, PEST multiplies it by its SCALE variable (-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, enter 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.

2.2.5 Parameter Change Limits

As has already been discussed, no parameter can be adjusted by PEST above its upper bound or below its lower bound. However, there is a further limit on parameter changes, determined by the amount by which a parameter is permitted to change in any one optimisation iteration.

If the model under PEST's control exhibits reasonably linear behaviour, the updated parameter set determined by equations 2.23, 2.24, and 2.26 will result in a lowering of the objective function. However if the model is highly nonlinear, the parameter upgrade vector $\beta\mathbf{u}$ may "overshoot" the objective function minimum, and the new value of Φ may actually be worse than the old one. This is because equations 2.23 and 2.24 are based on a linearity assumption which may not extend as far into parameter space from the current parameter estimates as the magnitude of the upgrade vector which they predict.

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 optimisation iteration. Such limits may be of two types, viz. "relative" and "factor". You must inform PEST, through the parameter variable PARCHGLIM on the PEST control file, which type of change limit applies to each adjustable parameter. Two other PEST input variables, RELPARMAX and FACPARMAX, provide the maximum allowed relative and factor changes for all relative-limited and factor-limited parameters, respectively. Values for these variables are supplied at the beginning of the inversion process. They can also be

altered part of the way through a PEST run if desired; see Section 5.6. Note that log-transformed parameters must be factor-limited.

Let f represent the user-defined maximum allowed parameter factor change for factor-limited parameters (ie. FACPARMAX); f must be greater than unity. Then if b_0 is the value of a particular factor-limited parameter at the beginning of an optimisation iteration, the value b of this same parameter at the beginning of the next optimisation iteration will lie between the limits

$$b_0/f \le b \le fb_0 \qquad (2.44a)$$

if b_0 is positive, and

$$fb_0 \le b \le b_0/f \qquad (2.44b)$$

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 (ie. RELPARMAX); r can be any positive number. Then if b_0 is the value of a particular relative-limited parameter at the beginning of an optimisation iteration, its value b at the beginning of the next optimisation iteration will be such that

$$\left| b - b_0 \right| / \left| b_0 \right| \le r \qquad (2.45)$$

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.

In implementing the conditions set by equations 2.44 and 2.45, PEST limits the magnitude of the parameter upgrade vector $\beta\mathbf{u}$ such that neither of these equations is violated. Naturally, if only one type of parameter change limit is featured in a current PEST run (ie. parameters are all factor-limited or are all relative-limited) only the pertinent one of these equations will need to be obeyed.

If, in the course of an optimisation run, PEST assigns to a parameter a value which is very small in comparison with its initial value, then either of equations 2.44 or 2.45 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 2.44 or equation 2.45 is obeyed for this one parameter. To circumvent this problem, PEST provides another input variable, FACORIG, which allows the user to limit the effect that an unduly low 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 2.44 and the denominator of equation 2.45 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. For more information on FACORIG see Section 4.2.2.

It should be noted that problems such as those described above incurred by parameters with low absolute values can also be prevented from occurring by providing such parameters with a suitable OFFSET value, accompanied by appropriate lower/upper bounds that prevent them from being assigned such troublesome values.

2.2.6 Damping of Parameter Changes

Parameter over-adjustment and any resulting oscillatory behaviour of the parameter estimation process is further mitigated by the "damping" of potentially oscillatory 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 $\beta\mathbf{u}$ has just been determined using equations 2.23, 2.24 and 2.26. Suppose, further, that this upgrade vector causes no parameter values to exceed their bounds, and that all parameter changes are within factor and relative 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 factor-limited parameters which are not log-transformed, define q_j for parameter j as

$$\begin{aligned} q_j &= \beta u_j / (f b_j - b_j) \quad \text{if } u_j \text{ and } b_j \text{ have the same sign, and} \\ q_j &= \beta u_j / (b_j - b_j / f) \quad \text{if } u_j \text{ and } b_j \text{ have the opposite sign} \end{aligned} \tag{2.46}$$

where b_j is the current value for 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 $\beta\mathbf{u}$ is greatest be parameter k, and let the element of $\beta\mathbf{u}$ pertaining to this parameter be βu_k. Let i_0, l_0, k_0, p_{0i}, q_{0l} and $\beta_0 u_{0k}$ 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 and s_3 such that

$$\begin{aligned} s_1 &= p_i / p_{0i} \quad \text{if } i = i_0; \\ s_1 &= 0 \quad \text{otherwise,} \end{aligned} \tag{2.47a}$$

$$\begin{aligned} s_2 &= q_l / q_{0l} \quad \text{if } l = l_0; \\ s_2 &= 0 \quad \text{otherwise, and} \end{aligned} \tag{2.47b}$$

$$\begin{aligned} s_3 &= \beta u_k / \beta_0 u_{0k} \quad \text{if } k = k_0; \\ s_3 &= 0 \quad \text{otherwise.} \end{aligned} \tag{2.47c}$$

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

$$\rho = (3 + s)/(3 + |s|) \quad \text{if } s \geq -1 \tag{2.48a}$$

$$\rho = 1/(2|s|) \quad \text{otherwise.} \tag{2.48b}$$

Then oscillatory behaviour of the parameter estimation process can be mitigated by defining a new parameter upgrade vector $\mathbf{v}$ by

$$\mathbf{v} = \rho\beta\mathbf{u} \qquad (2.49)$$

2.2.7 Temporary Holding of Insensitive Parameters

The possibility of a parameter estimation process running smoothly and efficiently decreases with the number of parameters being estimated. In highly parameterised problems some parameters are likely to be relatively insensitive in comparison with other parameters. As a result of their insensitivity, PEST may decide that large changes are required for their values if they are to make any contribution to reducing the objective function. However, as is explained in Section 2.2.5, limits are set on parameter changes. These limits are enforced in such a way that the magnitude (but not the direction) of the parameter upgrade vector is reduced (if necessary) such that no parameter transgresses these limits. The necessity for such limits has already been discussed.

If a parameter is particularly insensitive, it may dominate the parameter upgrade vector, ie. the magnitude of the change calculated by PEST for this parameter may be far greater than that calculated for any other parameter. When its change has been relative- or factor-limited in accordance with the user-supplied settings for RELPARMAX or FACPARMAX (and the magnitude of the parameter upgrade vector has thus been considerably reduced), other parameters (including far more sensitive ones) may not change much at all, with the result that at the end of the optimisation iteration the objective function may have been lowered very little. The same process may then be repeated on the next iteration on account of the same, or another, insensitive parameter. The result may be that convergence takes place intolerably slowly (or not at all), with a huge wastage of model runs.

This phenomenon can be avoided by temporarily holding troublesome parameters at their current value for an iteration or two. Such parameters are then not involved in the calculation of the parameter upgrade vector and hence do not get the chance to have an adverse impact on it. Offending parameters can be identified as those undergoing the maximum relative- or factor-limited changes during a particular optimisation iteration where this maximum change is equal to RELPARMAX or FACPARMAX, or as those parameters whose current sensitivity is very low, PEST recording information by which to make this assessment every time it calculates the Jacobian matrix.

PEST records the "composite sensitivity" of each parameter (ie. the magnitude of the column of the Jacobian matrix pertaining to that parameter modulated by the weight attached to each observation divided by the number of observations, or V_{ii}/m where V_{ii} is the inverse of S_{ii} defined in equation 2.22 and m is the number of observations - see equation 5.1), to a "parameter sensitivity file", this file being updated during every optimisation iteration. Those parameters with the lowest sensitivities are the most likely to cause trouble and hence the most likely candidates for being temporarily held if the parameter estimation process proceeds too slowly as a result of one or more parameters encountering their relative or factor limits, thus restricting alterations made to other parameters to ineffectual levels. See Section 5.6 for further details.

2.2.8 Components of the Objective Function

As has already been discussed, the objective function is calculated as the squared sum of weighted residuals (including prior information). If is often of interest to know what

contribution certain observations, or groups of observations, make to the objective function. This is possible through the use of "observation groups". Each observation, and each item of prior information, must be assigned to a group; the number and names of such groups are specified by the user.

The ability to calculate the contribution made by individual observations, or groups of observations to the objective function is useful in situations where the user wishes that different types of information contribute an approximately equal amount to the value of the objective function. This ensures that no observation groupings are either "drowned" by other information, or dominate the inversion process.

2.2.9 Termination Criteria

PEST updates parameters using equations derived on the basis of a linearity assumption which is not met if the model is nonlinear. Nevertheless, by iteratively updating the parameters in accordance with these equations as many times as is necessary, an optimal parameter set will mostly be obtained in the end. When working in parameter estimation mode the optimal set of parameters is that set for which the objective function is at its minimum.

PEST uses a number of different criteria to determine when to halt this iterative process. Note that only one of them (zero-valued objective function) is a guarantee that the objective function minimum has been obtained. In difficult circumstances, any of the other termination criteria could be satisfied when the objective function is well above its minimum and parameters are far from optimal. Nevertheless, in most cases these termination criteria do, indeed, signify convergence of the adjustable parameters to their optimal values. In any case, PEST has to stop executing sometime and each of the termination criteria described in this section provide as good a reason to stop as any. If these criteria are properly set through user-provided PEST input variables, you can be reasonably assured that when PEST terminates the parameter estimation process, either the optimal parameter set has been found or further PEST execution will not find it.

There are two indicators that either the objective function is at, or very close to, its minimum, or that further PEST execution is unlikely to get it there. The first is the behaviour of the objective function itself. If it has been reduced very little, or not at all, over a number of successive iterations, the time has come to cease execution. The exact criteria determining this kind of termination are set through PEST input variables PHIREDSTP, NPHISTP and NPHINORED. If the lowest NPHISTP Φ's achieved in all iterations carried out to date are within a distance of PHIREDSTP of each other relative to the lowest Φ achieved so far, or if NPHINORED iterations have elapsed since the lowest Φ was achieved, then PEST execution will cease.

The second indicator of either convergence to the objective function minimum, or of the unlikelihood of achieving it, is the behaviour of the adjustable parameters. If successive iterations are effecting little change in parameter values, there is probably little to gain in continuing with PEST execution. Input variables RELPARSTP and NRELPAR set the exact criterion; if the largest relative parameter change over the last NRELPAR iterations has been RELPARSTP or less, PEST will not proceed to the next iteration.

The input variable NOPTMAX sets an upper limit on the number of optimisation iterations which PEST carries out. PEST will terminate execution after NOPTMAX iterations, no matter what the current status of the objective function or of the parameter values.

Other termination criteria are set internally. As has already been mentioned, PEST will terminate the optimisation process if it calculates a parameter set for which the objective function is zero. Also, if the gradient of the objective function with respect to all parameters is zero, or if a zero-valued parameter upgrade vector is determined, or if all parameters are simultaneously at their limits and the parameter upgrade vector points out of bounds, PEST will take its deliberations no further (unless it is currently calculating derivatives using forward differences and the option to use central differences is available to it, in which case it will switch to the use of central differences for greater derivatives accuracy before moving on to the next iteration – see Section 2.3).

2.2.10 Operation in Predictive Analysis Mode

Most aspects of PEST's operation when undertaking predictive analysis are identical to its operation when undertaking parameter estimation, including the use of parameter bounds, relative and factor change limits, switching to the use of three-point derivatives calculation, prior information, the linking and fixing of parameters, the holding of parameters, logarithmic transformation, etc. All termination criteria that are used in parameter estimation mode also apply to PEST's use in predictive analysis mode. However, as discussed in Section 6.2.2, a number of extra termination criteria, applicable only to this mode of operation, are available.

As is explained in Section 6.1.5, if the initial parameter estimates supplied to PEST at the commencement of a predictive analysis run are a long way from optimum (ie. the initial objective function is far above Φ_0 of equation 2.28), PEST will work in parameter estimation mode until it is able to "sniff" the Φ_0 contour. The transition to predictive analysis mode as it approaches this contour is a gradual one, unseen by the user.

2.2.11 Operation in Regularisation Mode

Within each optimisation iteration PEST's task when working in regularisation mode is identical to its task when working in parameter estimation mode, ie. it must minimise an objective function using a linearised version of the model encapsulated in a Jacobian matrix. However just before calculating the parameter upgrade vector, PEST calculates the appropriate "regularisation weight factor" to use for that iteration. This is the factor by which all of the weights pertaining to regularisation information are multiplied (in accordance with equation 2.33) prior to formulating the overall objective function whose task it is for PEST to minimise on that iteration. As parameters shift and the Jacobian matrix changes (an outcome of the nonlinear nature of most models), the regularisation weight factor also changes. Hence it needs to be re-calculated during every optimisation iteration.

Use of PEST in regularisation mode is fully described in Chapter 7 of this manual. As is discussed in that Chapter, the user is required to supply a few extra control variables to govern PEST's operation in this mode. One of these is the "target measurement objective function" (ie. Φ_m^1 of equation 2.31). Other variables govern the procedure by which μ of equation 2.33 is calculated, and allow slight changes to be made to the criteria that govern

termination of a PEST run.

2.3 The Calculation of Derivatives

2.3.1 Forward and Central Differences

The ability to calculate the derivatives of all observations with respect to all adjustable parameters is fundamental to the Gauss-Marquardt-Levenberg method of parameter estimation; 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-generated observations 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 can be provided to it in the correct format. This is further described in Section 8 of this manual. Most of the discussion in the remainder of this Chapter assumes that PEST must calculate parameter derivatives itself.)

Accuracy in derivatives calculation is fundamental to PEST's success in optimising parameters. Experience has shown that the most common cause of PEST's failure to find the global minimum of the objective function in parameter space is the presence of roundoff errors incurred in the calculation of derivatives. Fortunately, on most occasions, this problem can be circumvented by a wise choice of those input variables which determine how PEST evaluates derivatives for a particular model.

The PEST input variables affecting derivatives calculation pertain to parameter "groups". In the PEST control file, each parameter must be assigned to such a parameter group. The assignment of derivative variables to groups, rather than to individual parameters, introduces savings in memory and complexity. 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 subdivided into zones of constant parameter value, and the parameters pertaining to all of these zones are being estimated, parameters of the same type for each such zone would normally belong 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.

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), 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.) 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 for every optimisation iteration, each optimisation 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 PEST's parameter estimation procedure.

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 (ie. 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 the model output file with great precision, because of either deficiencies in the model's numerical solution method, or because the model writes numbers to its output file using a limited number of significant figures.

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

2.3.2 Parameter Increments for Derivatives Calculation

Because of the importance of reliable 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 three-point 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 will suffer because of the effect of roundoff 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.

There are three PEST input variables, viz. INCTYP, DERINC and DERINCLB by which you can set the manner in which increments are calculated for the members of a particular parameter group. INCTYP determines the type of increment to use, for which there are three options, viz. "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 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 PEST group input 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.

The 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 or both; FORCEN can be designated as "always_2", "always_3" or "switch". If it is supplied as "always_2", derivatives calculation is through forward differences for all parameters within the group throughout the estimation process; if it is "always_3", central (ie. three-point) derivatives will be used for the entirety of the estimation process. However if it is "switch", PEST will commence the optimisation 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 optimisation iterations is less than the value contained in the PEST input variable PHIREDSWH.

Two group input variables pertain specifically to the calculation of derivatives using the central method, viz. variables DERINCMUL and DERMTHD. The latter variable must be 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.

The variable DERINCMUL contains the 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). Use of the higher-order interpolation scheme provided by the parabolic method may allow you to place parameter values, and hence model-generated observation values, farther apart for the calculation of derivatives; this may have the effect of increasing the degree of significance of the resulting differences involved in the derivative calculation. 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.

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

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 input 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 should be pointed out 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.

2.3.3 How to Obtain Derivatives You Can Trust

Reliability of derivatives calculation can suffer if the model which you are trying to parameterise does not write its outcomes to its output file 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 an optimisation on the basis of limited-precision model-generated observations, its ability to find an objective function minimum decreases as the precision of these 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 precision of model output.

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 maximum 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 judge that the solution is 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. 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 computation time.

If a numerical model of this type is to be used with PEST, *it is essential that any variables governing the numerical solution procedure be set in favour of precision over time*. Although the model run-time may be much greater as a result, it would be false economy to give reduced computation time 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 optimisation 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 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" in that the relationship between these observations and the model parameters may be "bumpy" rather than continuous. 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 central methods of derivatives calculation available in PEST, particularly the parabolic and best-fit methods. 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.

2.3.4 Model-Calculated Derivatives

As has been discussed above, the calculation of derivatives by finite differences is both time-consuming and numerically intensive. If a model can calculate derivatives of its outputs with respect to its adjustable parameters itself, use of these derivatives is normally extremely beneficial to the parameter estimation process. This is a result of the greater 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 based derivatives.

Chapter 8 of this manual describes the mechanism by 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 when calculating 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 8 for a discussion of both of these.

As is also described in Chapter 8 there will be some situations (especially those involving calibration and predictive analysis for complex models) 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.

2.4 Bibliography

2.4.1 Literature Cited in the Text

Cooley, R.L., 1983. Some new procedures for numerical solution of variably saturated flow problems. Water Resources Research, v19, no. 5, p1271-1285.

Cooley, R.L. and Naff, R.L., 1990. Regression modeling of ground-water flow: U.S. Geological Survey Techniques in Water-Resources Investigations, book 3, chap B4, 232p.

Cooley, R.L. and Vecchia, A.V., 1987. Calculation of nonlinear confidence and prediction intervals for ground-water flow models. Water Resources Bulletin. Vol. 23, No. 4, pp581-599.

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.

Hill, M.C., 1998. Methods and Guidelines for Effective Model Calibration. U.S. Geological Survey Water-Resources Investigations Report 98-4005.

Marquardt, D. W., 1963. An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society of Industrial and Applied Mathematics, v11, no. 2, p431-441.

Levenberg, K., 1944. A method for the solution of certain non-linear problems in least squares. Q. Appl. Math., v. 2, p164-168.

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 Resources Research, vol. 23, no. 7, pp1237-1250.

2.4.2 Some Further Reading

Bard, Jonathon, 1974. Nonlinear parameter estimation. Academic Press, NY. 341p.

Koch, K., 1988. Parameter Estimation and Hypothesis Testing in Linear Models. Springer-Verlag, Berlin. 377p.

Mikhail, E. M., 1976. Observations and Least Squares. IEP, NY. 497p.

Nash, J. C. and Walker-Smith, M., 1987. Nonlinear Parameter Estimation; an Integrated System in Basic. Marcel Dekker Inc., Monticello, NY. 493p.

3. The Model-PEST Interface

3.1 PEST Input Files

PEST requires three types of input file. These are:

- template files, one for each model input file on which parameters are identified,
- instruction files, one for each model output file on which model-generated observations are identified, and
- an input 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 in Chapter 4. Template files and instruction files can be written using a general-purpose text editor following the specifications set out in this chapter. Once built, they can be checked for correctness and consistency using the utility programs TEMPCHEK, INSCHEK and PESTCHEK; these programs are described in Chapter 10 of this manual.

Note that in this and other chapters of this manual, the word "observations" is used to denote those particular model outcomes for which there are corresponding laboratory or field data. For clarity, these numbers are often referred to as "model-generated observations" to distinguish them from their laboratory- or field-acquired counterparts which are referred to as "measurements" or "laboratory or field observations".

3.2 Template Files

3.2.1 Model Input Files

Whenever PEST runs a model, as it must do many times in the course of the optimisation process, 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 optimisation. PEST does not need to know about any of the other model input files.

PEST can only write parameters to ASCII (ie. 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. 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.

3.2.2 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 Example 3.1; 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 (ie. model) to estimate the properties for 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 Example 3.2.

```
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
```

Example 3.1 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
```

Example 3.2 A template file.

3.2.3 The Parameter Delimiter

As Example 3.2 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.

3.2.4 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) and on the PEST control file (where parameter initial values, lower and upper bounds and other information are provided). 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.

3.2.5 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, viz. from specified fields, or as a sequence of numbers, each of which may be of any length; in FORTRAN the latter method is often referred to as "free field" input. If the model uses the former method, then somewhere within the model program the format (ie. field specification) for data entry is defined for every number which must be read in this fashion.

The FORTRAN code of Example 3.3 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
```

Example 3.3 Formatted and free field input.

The relevant part of the input file may be as illustrated in Example 3.4.

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

Example 3.4 Numbers read using the code of Example 3.3

Notice how no whitespace or comma is needed between numbers which are read using a field specifier. The format statement labelled "100" in Example 3.3 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 optimising 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 Example 3.4 may then be as set out in Example 3.5. 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 Example 3.3. If the parameter space for any of these parameters were 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

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

Example 3.5 Fragment of a template file corresponding to Example 3.4

using less than the full 10 characters allowed by the model.)

Parameters D and E are treated very differently to parameters A, B and C. As Example 3.3 shows, the model simply expects two numbers in succession. If the spaces for parameters D and E appearing in Example 3.5 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 Examples 3.4 and 3.5, 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 Example 3.2 pertaining to the model input file of Example 3.1. 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 (eg. variable A in Example 3.4). 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.)

3.2.6 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; see Section 4.2.2 for details. 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 (eg. for specifiers such as "(F6.2)"), or enforce power-of-10 scaling (eg. 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 3.1 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.

Table 3.1 Representations of the number 12345.67

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	**	**

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.

3.2.7 Multiple Occurrences of the Same Parameter

Large numerical models which calculate the variation of some scalar or vector 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 ground water 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 optimise 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 (see Chapter 4) 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.

3.2.8 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 Chapter 10. 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.

3.3 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, viz. those numbers for which corresponding field or laboratory data are available and for which the discrepancy between model output and measured values must be reduced to a minimum in the weighted least squares sense. These particular model-generated numbers are referred to as "observations" or "model-generated observations" in the discussion which follows (in order to distinguish them from those model outcomes which are not used in the parameter estimation process), while the field or laboratory data to which they are individually matched 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.

3.3.1 Precision in Model Output Files

As was discussed in the previous chapter, if there are any model input variables which allow you to vary the precision with which its output data are written, they should be adjusted for maximum output precision. Unlike parameter values, for which precision is important but not essential, precision in the representation of model-generated observations is crucial. The Gauss-Marquardt-Levenberg method of nonlinear parameter estimation, upon which the PEST algorithm is based, requires that the derivative of each observation with respect to each parameter be evaluated once for every optimisation iteration. PEST calculates these derivatives using the finite difference technique or one of its three-point variants. In all cases, the derivative value depends on the difference between two or three observations calculated on the basis of incrementally-varied parameter values. Unless the observations are represented with maximum precision, this is a recipe for numerical disaster.

3.3.2 How PEST Reads a Model Output File

PEST must be instructed on how to read a model output file and identify model-generated observations. For the method to work, model output files containing observations 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 read the ASCII file for the observations it needs. Note that, as described in Section 4.2.8, when PEST runs a model, this "model" can actually consist of a number of programs run in succession.

Unfortunately, observations 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 stress regime in an aircraft wing 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 milliseconds, you may instruct PEST to read its output file looking for the marker

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

A particular outcome for which you have a corresponding experimental 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.

3.3.3 An Example Instruction File

Example 3.6 shows an output file written by the model whose input file appears in Example 3.1. Suppose that we wish to estimate the parameters appearing in the template file of Example 3.2 (ie. 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 Example 3.6. An appropriate instruction file is shown in Example 3.7.

```
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
```

Example 3.6 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
```

Example 3.7 A PEST instruction file.

3.3.4 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, !, [,], (,), :, 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.

3.3.5 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. Like a parameter name, an observation name must be twelve characters or less in length. These twelve 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 Example 3.6, 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; see the next chapter for further details.

There is one observation name, however, to which these rules do not apply, viz. 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 mechanism for model output file navigation.

3.3.6 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, viz. 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 Example 3.7, 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 Example 3.8 (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 Example 3.9 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
```

Example 3.8 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
     .
     .
```

Example 3.9 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 for a line advance it 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 Example 3.6 model-calculated apparent resistivities are written on subsequent 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 Example 3.7.

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.

Example 3.10 shows an extract from a model output file while Example 3.11 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
            .
            .
```

Example 3.10 Extract from a model output file.

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

Example 3.11 Extract from an instruction file.

A useful feature of the secondary marker is illustrated in Examples 3.12 and 3.13 of 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 Example 3.12. However, when it does not find the string "STRAIN" on the same line it recommences 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
            .
            .
```

Example 3.12 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 abort execution with an appropriate error message.

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

Example 3.13 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 (ie. 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:", ie. 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, ie. 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 (ie. 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".

Example 3.14 shows how the numbers listed in the third solution vector of Example 3.8 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
     .
     .
```

Example 3.14 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 in the course of the many model runs that will be required for PEST to optimise the model's parameter set; 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 will be

```
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

Example 3.9 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, 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, viz. 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 Examples 3.6 and 3.7, 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 Example 3.7. 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

Examples 3.11 and 3.13 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 Example 3.15. 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 Example 3.16.

```
            .
            .
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
            .
            .
```

Example 3.15 Extract from a model output file.

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

Example 3.16 Extract from an instruction file.

A primary marker is used to move the PEST cursor to the first of the lines shown in Example 3.15. 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 (see the next chapter). 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 (viz. *%*) 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 will 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 observation number 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 the "=", viz. 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.

3.3.7 Making an Instruction File

An instruction file can be built using a text editor. This is particularly easy in the WINDOWS environment where you can open two command-line windows, one to view a model output file, and the other to write the instruction file. If the viewing program is a text editor which displays cursor line and column numbers, the job is even easier; note that the text editor, EDIT, provides these numbers. Furthermore, with the help of the WINDOWS clipboard facility, you can easily copy markers from a model output file to an instruction file using the mouse.

You must always exercise caution 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 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. In this case you should interrupt PEST execution, asking PEST to terminate execution with a statistics printout (see Chapter 5). Included in this printout are the current model-generated observation values; if PEST is reading the wrong number it will then become apparent.

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 (see the next chapter) 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. See Chapter 10 of this manual for more details.

4. The PEST Control File

4.1 The Role of the PEST Control File

Once all the 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"; PEST uses this same filename base for the files which it generates in the course of its run. Thus, for example, if you name the PEST control file *calib.pst*, PEST will generate files *calib.rec* (the run record file), *calib.par* (best parameter values achieved), *calib.rst* (restart information stored at the beginning of each optimisation iteration), *calib.jac* (the Jacobian matrix for a possible restart), *calib.jst* (the same file from the previous optimisation iteration), *calib.jco* (the Jacobian matrix pertaining to best parameters for access by the JACWRIT utility), *calib.sen* (parameter sensitivities), *calib.seo* (observation sensitivities) and *calib.res* (tabulated observation residuals). User-supplied files *calib.rmf* (Parallel PEST run management file) and *calib.hld* (the "parameter hold file") are possible PEST input files.

Many of the data items in the PEST control file are used to "tune" PEST's operation to the case in hand; such items include parameter change limits, parameter transformation types, termination criteria etc. As is further discussed in Chapter 5, before using PEST on a real-world problem, you may wish to use it to estimate parameters for a case where your "field" data are, in fact, model-generated; in this way you know the answers that PEST should achieve. Through a careful examination of the PEST run record file, and perhaps a little experimentation with PEST input variables, you should be able to determine what PEST settings are best for your particular problem. (You can also test whether the observation set that you provide affords the determination of a unique parameter set.)

The PEST control file can be built in one of two ways. It can be easily prepared using a text editor following the directions provided in this chapter. Alternatively, you can use the PEST utility, PESTGEN, to generate a PEST control file for your current case using default input variables; this file can then be modified as you see fit using a text editor. In either case the PEST control file can be checked for correctness and consistency using the utility program PESTCHEK.

Note also that some of the programs of the PEST Ground Water and Surface Water Utilities can be used to write a PEST control file.

4.2 Construction Details

4.2.1 The Structure of the PEST Control File

The PEST control file consists of integer, real and character variables. Its construction details are set out in Example 4.1, where variables are referenced by name. A sample PEST control file is provided in Example 4.2. Note that the PEST control file demonstrated in these two

examples pertains to the use of PEST in "parameter estimation mode" (the most usual case of PEST usage). Use of PEST in "predictive analysis mode" is described in Chapter 6 while use of PEST in "regularisation mode" is described in Chapter 7. Note also that whenever PEST is upgraded and additional variables are required in the PEST control file, newer versions of PEST will always be capable of reading old versions of the PEST control file; default values will simply be assigned to missing variables.

```
pcf
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP
NTPLFLE NINSFLE PRECIS DPOINT NUMCOM JACFILE MESSFILE
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM
RELPARMAX FACPARMAX FACORIG
PHIREDSWH
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR
ICOV ICOR IEIG
* parameter groups
PARGPNME INCTYP DERINC DERINCLB FORCEN DERINCMUL DERMTHD
(one such line for each of the NPARGP parameter groups)
* parameter data
PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM
(one such line for each of the NPAR parameters)
PARNME PARTIED
(one such line for each tied parameter)
* observation groups
OBGNME
(one such line for each observation group)
* observation data
OBSNME OBSVAL WEIGHT OBGNME
(one such line for each of the NOBS observations)
* model command line
write the command which PEST must use to run the model
* model input/output
TEMPFLE INFLE
(one such line for each model input file containing parameters)
INSFLE OUTFLE
(one such line for each model output file containing observations)
* prior information
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of the NPRIOR articles of prior information)
```

Example 4.1 Construction details of the PEST control file.

```
pcf
* control data
restart estimation
5 19 2 2 4
1 1 single point 1  0  0
5.0 2.0 0.4 0.03 10
3.0 3.0 1.0e-3
.1
30 .01 3 3 .01 3
1 1 1
* parameter groups
ro relative .001 .00001 switch 2.0 parabolic
h  relative .001 .00001 switch 2.0 parabolic
* parameter data
ro1  fixed factor  0.5   .1  10     none 1.0 0.0 1
ro2  log   factor  5.0   .1  10     ro   1.0 0.0 1
ro3  tied  factor  0.5   .1  10     ro   1.0 0.0 1
h1   none  factor  2.0   .05 100    h    1.0 0.0 1
h2   log   factor  5.0   .05 100    h    1.0 0.0 1
ro3  ro2
* observation groups
group_1
group_2
group_3
group_4
* observation data
ar1    1.21038  1.0 group_1
ar2    1.51208  1.0 group_1
ar3    2.07204  1.0 group_1
ar4    2.94056  1.0 group_1
ar5    4.15787  1.0 group_1
ar6    5.77620  1.0 group_1
ar7    7.78940  1.0 group_2
ar8    9.99743  1.0 group_2
ar9    11.8307  1.0 group_2
ar10    12.3194  1.0 group_2
ar11    10.6003  1.0 group_2
ar12    7.00419  1.0 group_2
ar13    3.44391  1.0 group_2
ar14    1.58279  1.0 group_2
ar15    1.10380  1.0 group_3
ar16    1.03086  1.0 group_3
ar17    1.01318  1.0 group_3
ar18    1.00593  1.0 group_3
ar19    1.00272  1.0 group_3
* model command line
ves
* model input/output
ves.tp1 ves.inp
ves.ins ves.out
* prior information
pi1  1.0 * h1 = 2.0 3.0 group_4
pi2  1.0 * log(ro2) + 1.0 * log(h2) = 2.6026 2.0 group_4
```

Example 4.2 A PEST control file.

A PEST control file must begin with the letters “pcf” for “PEST control file”. Scattered through the file are a number of section headers. These headers always follow the same format, viz. an asterisk followed by a space followed by text. When preparing a PEST control

file, these headers must be written exactly as set out in Examples 4.1 and 4.2; however if there is no prior information, the "prior information" header can be omitted.

On each line of the PEST control file, variables must be separated from each other by at least one space. Real numbers can be supplied with the minimum precision necessary to represent their value; the decimal point does not need to be included if it is redundant. If exponentiation is required, this can be accomplished with either the "d" or "e" symbol; note, however, that all real numbers are stored internally by PEST as double precision numbers.

The data which must reside in the PEST control file is now discussed in detail section by section. Refer to Example 4.1 for the location within the PEST control file of each input variable discussed below.

4.2.2 Control Data

The data provided in the "control data" section of the PEST control file are used to set internal array dimensions, tune the optimisation algorithm to the problem at hand, and set some data output options.

RSTFLE

This character variable must be assigned one of two possible values, viz. "restart" or "norestart". (Note that for this, and other character variables in the PEST control file, PEST is case insensitive.) If it takes the value "restart", PEST will dump the contents of many of its data arrays to a binary file (named *case.rst* where *case* is the current case name) at the beginning of each optimisation iteration; this allows PEST to be restarted later if execution is prematurely terminated. If subsequent PEST execution is initiated using the "/r" command line switch (see Section 5.4.2), it will recommence execution at the beginning of the iteration during which it was interrupted. PEST will also dump the Jacobian matrix to a binary file named *case.jac* on every occasion of this matrix being filled. This allows re-commencement of execution with the "/j" switch which, as is explained in Section 5.6, may be useful if it is desired that PEST re-calculate the parameter upgrade vector with certain recalcitrant parameters temporarily held fixed.

If the RSTFLE variable is set to "norestart", PEST will not intermittently dump its array or Jacobian data; hence a later re-commencement of execution after premature termination is impossible.

PESTMODE

This character variable must be supplied as either "estimation", "prediction" or "regularisation". If set to "estimation" PEST will run in parameter estimation mode (its traditional mode of operation); if set to "prediction" PEST will run in predictive analysis mode; if set to "regularisation" PEST will run in regularisation mode.

If PEST is run in predictive analysis mode, you must ensure that the PEST control file contains a "predictive analysis" section. You must also ensure that there are at least two observation groups, one of which is named "predict", and that the "predict" group has just one observation. See Chapter 6 for further details.

If PEST is run in regularisation mode you must ensure that the PEST control file contains a "regularisation" section. You must also ensure that there are at least two observation groups, one of which is named "regul". See Chapter 7 for further details.

Note that if PESTMODE is supplied as "estimation" there is no need to include a "predictive analysis" section or a "regularisation" section in the PEST control file.

NPAR

This is the total number of parameters used for 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 used in the current case. Note that, when counting the number of observations to evaluate NOBS, any dummy observations (see Chapter 3) that may be referenced in one or a number of instruction files are ignored.

NPARGP

This is the number of parameter groups; parameter groups are discussed in detail below. NPARGP is an integer variable.

NPRIOR

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

In general, 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 the PEST "normal" matrix of equation 2.16 will not be positive definite (in fact it will be singular) and a unique solution to the parameter estimation problem will not achievable. Nevertheless PEST will probably still determine a parameter vector which minimises the objective function, using the Marquardt parameter to make the normal matrix positive definite; see equation 2.23. However, a parameter vector determined in this way is not unique; furthermore, PEST will be unable to determine the parameter covariance matrix and to calculate parameter uncertainty levels because the Marquardt lambda is not added to the normal matrix prior to the determination of these quantities.

NOBSGP

NOBSGP, another integer variable, is the number of observation groups used in the current case. 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 objective function it also evaluates the contribution made to the objective function by the observations and prior information equations belonging to each group.

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, ie. 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 treat 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 uses in the determination of the objective function. (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 treated 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; ie. 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 3.2.6.

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”, PEST will ensure that the decimal point is always present. See Section 3.2.6.

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 8. For normal operation these should be set at 1, 0 and 0 respectively.

RLAMBDA1

This real variable is the initial Marquardt lambda. As discussed in Section 2.1.7, PEST attempts parameter improvement using a number of different Marquardt lambdas during any

one optimisation iteration; however, in the course of the overall parameter estimation process, the Marquardt lambda generally gets smaller. An initial value of 1.0 to 10.0 is appropriate for most models, though if PEST complains that the normal matrix is not positive definite, you will need to provide a higher initial Marquardt lambda.

As explained in Section 2.1.7, for high values of the Marquardt lambda the parameter estimation process approximates the steepest-descent method of optimisation. While the latter method is inefficient and slow if used for the entirety of the optimisation process, it often helps in getting the process started, especially if initial parameter estimates are poor.

The Marquardt lambda used by PEST is subject to user-alteration midway through the optimisation process. See Section 5.6 for further details.

RLAMFAC

RLAMFAC, a real variable, is the factor by which the Marquardt lambda is adjusted; see Section 2.1.7. RLAMFAC must be greater than 1.0. 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 normal matrix is not positive definite or if a reduction in lambda does not lower the objective function, PEST has no choice but to increase lambda.

PHIRATSUF

During any one optimisation iteration, 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 optimisation iteration. Thus PEST will commence iteration i+1 if, at any stage during iteration i

$$\Phi_i^j/\Phi_{i\text{-}1} \leq \text{PHIRATSUF} \tag{4.1}$$

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

PHIRATSUF (which stands for "phi ratio sufficient") is a real variable for which a value of 0.3 is often 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 linear approximation upon which the optimisation equations of Chapter 2 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 parameter estimation 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 are tested,

PEST must use some other criterion in deciding when it should move on to the next optimisation 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 one optimisation 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 optimisation iteration than to continue testing the effect of new Marquardt lambdas. Thus if

$$(\Phi_i^{j-1} - \Phi_i^{j})/\Phi_i^{j-1} \leq \text{PHIREDLAM} \qquad (4.2)$$

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 often around 0.01. If it is set too large, the criterion for moving on to the next optimisation 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 optimisation iteration when it would be better off starting on a new iteration.

NUMLAM

This integer variable places an upper limit on the number of lambdas that PEST can test during any one optimisation iteration. It should normally be set between 5 and 10. For cases where parameters are being adjusted near their upper or lower limits, and for which some parameters are consequently being frozen (thus reducing the dimension of the problem in parameter space) experience has shown that a value closer to 10 may be more appropriate than one closer to 5; this gives PEST a greater chance of adjusting to the reduced problem dimension as parameters are frozen.

RELPARMAX and FACPARMAX

As was explained in Section 2.2.5, there should be some limit placed on the amount by which parameter values are allowed to change in any one optimisation iteration. If there is no limit, parameter adjustments could regularly "overshoot" their optimal values, causing a prolongation of the estimation process at best, and instability with consequential estimation failure at worst; the dangers are greatest for highly nonlinear problems.

PEST provides two input variables which can be used to limit parameter adjustments; these are RELPARMAX and FACPARMAX, both real variables. 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. Any particular parameter can be subject to only one of these constraints; ie. a particular parameter must be either relative-limited or factor-limited in its adjustments. Parameters are denoted as either relative-limited or factor-limited through the character variable PARCHGLIM supplied for each parameter; see below.

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

$$(b_{i-1} - b_i)/b_{i-1} \qquad (4.3)$$

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 optimisation iterations i-1 and i is defined as

$$\begin{aligned} &b_{i-1}/b_i && \text{if } |b_{i-1}| > |b_i|, \text{ or} \\ &b_i/b_{i-1} && \text{if } |b_i| > |b_{i-1}| \end{aligned} \qquad (4.4)$$

If parameter b is factor-limited, this factor change (which either equals or exceeds unity according to equation 4.4) 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.

Whether a parameter should be relative-limited or factor-limited depends on the parameter. However you should note that a parameter can be reduced from its current value right down to zero for a relative change of only 1; as described in Section 2.2.5 it may then take many iterations to re-adjust upwards, this causing serious inefficiencies in 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 optimisation process, it must be relative-limited; furthermore RELPARMAX must be set at greater than unity or the change of sign will be impossible. Thus the utility program PESTCHEK (see Chapter 10) will not allow you to declare a parameter as factor-limited, or as relative-limited with the 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 can vary enormously between cases. For highly non-linear problems, these values are best set low. If they are set too low, however, the estimation 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 each optimisation 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 an insensitive parameter should be held at its current value - see Section 5.6).

If you are unsure of how to set these parameters, a value of 5 for each of them is often suitable. In cases of extreme nonlinearity, be prepared to set them much lower. Note, however, 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.

Values assigned to RELPARMAX and FACPARMAX can be adjusted in the course of the optimisation process through the user-intervention functionality discussed in Section 5.6.

FACORIG

If, in the course of the estimation 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 4.3 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 4.3. For factor-limited parameters, a similar modification to equation 4.4 applies. 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.

PHIREDSWH

The derivatives of observations with respect to parameters can be calculated using either forward differences (involving two parameter-observation pairs) or one of the variants of the central method (involving three parameter-observation pairs) described in Section 2.3. As discussed below, you must inform PEST through the variables FORCEN and DERMTHD which method is to be used for the parameters belonging to each parameter group.

Using the variable FORCEN, you may wish to decree that, for a particular parameter group, derivatives will first be calculated using the forward difference method and later, when PEST is faltering in its attempts to reduce the objective function, calculated using one of the central methods. Alternatively, you may direct that no such switching take place, the forward or central method being used at all times for the parameters belonging to a particular group. In the former case you must provide PEST with a means of judging when to make the switch; this is the role of the real variable PHIREDSWH.

If the relative reduction in the objective function between successive optimisation iterations is less than PHIREDSWH, PEST will make the switch to three-point derivatives calculation for those parameter groups for which the character variable FORCEN has the value "switch"; thus if, for the *i'*th iteration

$$(\Phi_{i-1} - \Phi_i)/\Phi_{i-1} \leq \text{PHIREDSWH} \qquad (4.5)$$

(where Φ_i is the objective function calculated on the basis of the upgraded parameter set determined in the *i'*th iteration), PEST will use central derivatives in iteration i+1 (and all succeeding iterations) for all parameter groups for which FORCEN is "switch". A value of 0.1 is often suitable for PHIREDSWH. If it is set too high, PEST may make the switch to three-point 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 estimation process. If PHIREDSWH is set too low, PEST may waste an optimisation 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 central 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 optimisation iterations.

NOPTMAX

The input variables on the ninth line of the PEST control file set the termination criteria for the parameter estimation process. These are the criteria by which PEST judges that the optimisation process has been taken as far as it can go. These should be set such that either parameter convergence to the optimal parameter set has been achieved, or it has become obvious that continued PEST execution will not bear any fruits.

The first number required on this line is the integer variable NOPTMAX. This sets the maximum number of optimisation iterations that PEST is permitted to undertake on a particular parameter estimation run. If you want to ensure that PEST termination is triggered by other criteria, more indicative of parameter convergence to an optimal set or of the futility of further processing, you should set this variable very high. A value of 20 to 30 is often appropriate.

Two possible settings for NOPTMAX have special significance. If NOPTMAX is set to 0, PEST will not calculate the Jacobian matrix. Instead it will terminate execution after just one model run. This setting can thus be used when you wish to calculate the objective function corresponding to a particular parameter set and/or to inspect observation residuals corresponding to that parameter set.

If NOPTMAX is set to –1, PEST will terminate execution immediately after it has calculated the Jacobian matrix for the first time. The parameter covariance, correlation coefficient and eigenvector matrices will be written to the run record file, and parameter sensitivities will be written to the sensitivity file; these are based on the initial parameter set supplied 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

$$(\Phi_i - \Phi_{min})/\Phi_i \leq \text{PHIREDSTP} \qquad (4.6)$$

(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 optimisation process is at an end.

For many cases 0.01 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 optimisation iterations to ensure that no parameter value overshoots its bound. The result may be smaller reductions in the objective function than would otherwise occur. 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 optimisation iterations is less than RELPARSTP over NRELPAR successive iterations, PEST will cease execution. The relative parameter change between optimisation iterations for any parameter is calculated using equation 4.3. PEST evaluates this change for all adjustable parameters at the end of each optimisation iteration, 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 0.01 is often suitable. NRELPAR is an integer variable; a value of 2 or 3 is normally satisfactory.

ICOV, ICOR and IEIG

As is explained in Section 5.3.5, at the end of each optimisation iteration PEST writes a "matrix file" containing the covariance and correlation coefficient matrices, as well as the eigenvectors and eigenvalues of the covariance matrix based on current parameter values. The settings of the ICOV, ICOR and IEIG variables determine which (if any) of these data are recorded on the matrix file. A setting of 1 for each of these variables will result in the corresponding data being recorded on the matrix file. On the other hand, a setting of 0 will result in the corresponding data not being recorded. If all of these variables are set to zero none of this data will be recorded. Where a large number of parameters are being estimated (as might happen, for example, when PEST is being used in regularisation mode), setting all of these variables to 0 may result in some savings in computation time, for there is then no need for PEST to calculate the covariance matrix until the end of the parameter estimation process when these matrices and eigenvalues/eigenvectors are recorded on the run record file (irrespective of the settings of ICOV, ICOR and IEIG).

4.2.3 Parameter Groups

Each adjustable parameter (ie. each parameter which is neither fixed nor tied) must belong to a parameter group; the group to which each such parameter belongs is supplied through the parameter input variable PARGP (see below). Each parameter group must possess a unique name of twelve characters or less.

The PEST input variables that define how derivatives are calculated pertain to parameter groups rather than to individual parameters. Thus derivative data does not need to be entered individually for each parameter; however, if you wish, you can define a group for every parameter and set the derivative variables for each parameter separately. In many cases parameters fall neatly into separate groups which can be treated similarly in terms of calculating derivatives. For example in Example 4.2, which is a PEST control file for a case involving the calculation of surface-measured apparent resistivities from layered-half-space properties, the layer resistivities can be assembled into a single group, as can the layer thicknesses.

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). Alternatively, fixed or tied parameters can be assigned to the dummy group "none". If any group name other than "none" is cited for any parameter input variable PARGP 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. Note that an adjustable parameter cannot be assigned to the dummy group "none".

As Example 4.1 shows, one line of data must be supplied for each parameter group. Seven entries are required in each such line; the requirements for these entries are now discussed in detail.

PARGPNME

This is the parameter group name; all PEST names (viz. parameter, observation, prior information and group names) are case-insensitive and must be a maximum of twelve characters in length. If derivative data is provided for a group named by PARGPNME, 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.

Note that derivative variables cannot be defined for the group "none" as this is a dummy group name, reserved for fixed and tied parameters for which no derivatives information is required.

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. See Section 2.3 for a full discussion of the methods used by PEST to calculate parameter derivatives.

Thus if INCTYP is "relative" and DERINC is 0.01 (a suitable value in many cases), the increment for each group member for each optimisation 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 optimisation 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 optimisation 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 0.01 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 roundoff 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 will be the same for all group members, and is provided as the input variable DERINCLB. 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, of whether both alternatives are used in the course of an optimisation run. It must assume one of the values "always_2", "always_3" or "switch".

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; as explained in Section 2.3, 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 accuracy and this will probably have a beneficial effect on PEST's performance. If FORCEN is set to "switch", derivatives calculation for all adjustable group members will begin using the forward difference method, switching to the central method for the remainder of the estimation process on the iteration after the relative objective function reduction between successive optimisation iterations is less than PHIREDSWH, a value for which is supplied 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 optimisation 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 accuracy as possible, especially if parameters are highly correlated and the normal matrix thus approaches singularity.

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. In some cases it may be desirable to increase the value of the increment for this process over 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 "relative" or "rel_to_max" increment types, or holds the parameter increment itself, as it does for "absolute" increment types.

As discussed in Section 2.3.3, for many models the relationship between observations and parameters, while being in theory continuously differentiable, is often "granular" when examined under the microscope, this granularity being a by-product of the numerical solution scheme used by the model. In such cases the use of parameter increments which are too small may lead to highly inaccurate derivatives calculation, especially if the two or three sets of parameter-observation pairs used in a particular derivative calculation are on the same side of a "bump" in the parameter-observation relationship. Parameter increments must be chosen large enough to cope with model output granularity of this type. But increasing parameter increments beyond a certain amount diminishes the extent to which finite differences can approximate derivatives, the definition of the derivative being the limit of the finite difference as the increment approaches zero. However the deterioration in the derivative approximation as increments are increased is normally much greater for the forward difference method than for any of the central methods (particularly the "parabolic" option). Hence, the use of one of the central methods with an enhanced derivative increment may allow you to calculate derivatives in an otherwise hostile modelling environment.

Whenever the central method is employed for derivatives calculation, DERINC is multiplied by DERINCMUL, no matter whether INCTYP is "absolute", "relative" or "rel_to_max", and whether FORCEN is "always_3" or "switch". If you do not wish the increment to be increased, you must provide DERINCMUL with a value of 1.0. Alternatively, if for some reason you wish the increment to be reduced if three-point derivatives calculation is employed, you should provide DERINCMUL with a value of less than 1.0. Experience shows that a value between 1.0 and 2.0 is usually satisfactory.

DERMTHD

There are three variants of the central (ie. three-point) method of derivatives calculation; each method is described in Section 2.3. 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 be supplied as "parabolic", "best_fit" or "outside_pts". If FORCEN is "always_2", you must still provide one of these three legal values for DERMTHD; however for such a parameter group, the value of DERMTHD has no bearing on derivatives calculation for the member parameters.

4.2.4 Parameter Data - First Part

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 (viz. 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; refer to Example 4.1 for the arrangement on the PEST control file of the PEST input variables discussed below.

PARNME

This 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, viz. "none", "log", "fixed" or "tied".

If you wish that a parameter be log-transformed throughout the estimation process, the value "log" must be provided. As discussed in Section 2.2.1, logarithmic transformation of some parameters may have a profound affect on the success of the parameter estimation process. If a parameter is log-transformed PEST optimises 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 covariance, correlation coefficient and eigenvector matrices computed by PEST, 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 parameter estimation 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 parameter estimation run and an unsuccessful one. This is because, in many cases, the linearity approximation on which each PEST optimisation iteration is based holds better when certain parameters are log-transformed. However caution must be exercised when designating parameters as log-transformed. A parameter which can become zero or negative in the course of the parameter estimation process must not be log-transformed; hence if a parameter's lower bound is zero or less, PEST will disallow logarithmic transformation for 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. See Section 2.2.4 and the discussion on the SCALE and OFFSET variables below.)

If a parameter is fixed, taking no part in the optimisation 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 takes only a limited role in the estimation 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 parameter estimation 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 parameter estimation problem will benefit from a more complex parameter transformation type than logarithmic, this can be accomplished using the parameter preprocessor PAR2PAR; see Section 10.7 for further details.

PARCHGLIM

This character variable is used to designate whether an adjustable parameter is relative-limited or factor-limited; see Section 2.2.5 and the discussion of the input variables RELPARMAX and FACPARMAX above. PARCHGLIM must be provided with one of two possible values, viz. "relative" or "factor". For tied or fixed parameters this variable has no significance.

PARVAL1

PARVAL1, a real variable, is a parameter's initial value. For a fixed parameter, this value remains invariant during the optimisation process. For a tied parameter, the ratio of PARVAL1 to the parent parameter's PARVAL1 sets the ratio between these two parameters to be maintained throughout the optimisation 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 optimisation process.

To enhance optimisation efficiency, you should choose an initial parameter value which is close to what you think will be the parameter's optimised value. However you should note the following repercussions of choosing an initial parameter value of zero.

- A parameter cannot be subject to change limits (see the discussion on RELPARMAX and FACPARMAX) during the first optimisation iteration if its value at the start of that iteration is zero. Furthermore FACORIG cannot be used to modify the action of RELPARMAX and FACPARMAX for a particular parameter throughout the optimisation process, if that parameter's original value is zero.

- A relative increment for derivatives calculation cannot be evaluated during the first iteration for a parameter whose initial value is zero. If the parameter belongs to a group for which derivatives are, in fact, calculated 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 bounds 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 tied and parent parameters.)

PARGP

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

SCALE and OFFSET

Just before a parameter value is written to a model input file (be it for initial determination of the objective function, derivatives calculation or parameter upgrade), 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 estimation process; in fact they can "conceal" from PEST the true value of a parameter as seen by the model, PEST optimising, instead, the parameter b_p where

$$b_p = (b_m - o)/s \qquad (4.7)$$

Here b_p is the parameter optimised by PEST, b_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, enter the SCALE as 1.0 and the OFFSET as 0.0.

DERCOM

Unless using PEST's external derivatives functionality (see Chapter 8), this variable should be set to 1.

4.2.5 Parameter Data - 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 was 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, ie. 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.

4.2.6 Observation Groups

In the "observation groups" section of the PEST control file a name is supplied for every observation group. Like all other names used by PEST, observation group names must be of 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.

Observation group names are written one to a line. NOBSGP such names must be provided, where NOBSGP is listed on the fourth line 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 one of these group names must be "regul".

4.2.7 Observation Data

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 discussed below; refer to Example 4.1 for the arrangement of these items.

OBSNME

This is a character variable containing the observation name. As discussed in Section 3.3.5, an observation name must be twelve characters or less in length. Observation names are case-insensitive, but must be unique to each observation.

OBSVAL

OBSVAL, a real variable, is the field or laboratory measurement corresponding to a model-generated observation. It is PEST's role to minimise the difference between this number and the corresponding model-calculated number (the difference being referred to as the "residual") over all observations by adjusting parameter values until the sum of squared weighted residuals (ie. the objective function) is at a minimum.

WEIGHT

This is the weight attached to each residual in the calculation of the objective function. The manner in which weights are used in the parameter estimation process is discussed in Section 2.1.2. An observation weight can be zero if you wish (meaning that the observation takes no part in the calculation of the objective function), but it must not be negative.

If observations are all of the same type, weights can be used to discriminate between field or

laboratory measurements which you can "trust" and those with whom a greater margin of uncertainty is associated; the trustworthy measurements should be given a greater weight. Weights should, in general, be inversely proportional to measurement standard deviations.

If observations are of different types, weights are vital in setting the relative importance of each measurement type in the overall parameter estimation process. For example, a ground water model simulating pollution plume growth and decay within an aquifer may produce outputs of ground water head and pollutant concentration. Field measurements of both of these quantities may be available over a certain time period. If both sets of measurements are to be used in the model calibration process they must be properly weighted with respect to each other. Head measurements may be expressed in meters and pollutant concentrations may be expressed in meq/l. Heads may be of the order of tens of meters, with model-to-measurement discrepancies of up to 0.1 m being tolerable; however pollutant concentrations may be of the order of 10^{-4}meq/l, with model-to-measurement discrepancies of 0.1×10^{-4}meq/l being tolerable. In such a case the weights for the concentration measurements should be a factor of 10^4 greater than those for the head measurements so that both sets of measurements are equally effective in determining model parameters.

Some parameter estimation packages offer a "log least squares" option whereby the objective function is calculated as the sum of squared deviations between the logarithms of the measurements and the logarithms of their respective model-generated counterparts. Note that, provided the linearity assumption upon which the estimation process is based is reasonably well met, it can be shown that the same effect can be achieved by providing a set of weights in which each weight is inversely proportional to the measurement to which it pertains (provided all measurements are of the same sign).

OBGNME

OBGNME is the name of the observation group to which the observation is assigned. When recording the objective function value on the run record file, PEST lists the contribution made to the objective function by each observation group. It is good practice to assign observations of different type to different observation groups. In this way the user is in a position to adjust observation weights in order that one measurement type does not dominate over another in the inversion process by virtue of a vastly greater contribution to the objective function.

The observation group name supplied here must be one of the group names listed in the "observation groups" section of the PEST control file.

4.2.8 Model Command Line

This section of the PEST control file supplies the command which PEST must use to run the model. The command line may be simply the name of an executable file, or it may be the name of a batch 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 either the model must be in the current directory, its full path must be provided, or the PATH environment variable must include the directory in which the executable or batch file is situated.

Consider the case of a finite difference model for the stress field surrounding a tunnel. The input file 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, viz. the elastic properties of the zones into which the model domain has been subdivided. In this case it may be wise to run the preprocessor prior to every model run. This can be accomplished by including the commands to run both programs in a batch file called by PEST as the model; hence PEST can now write input files for the preprocessor rather than for the model itself.

Similarly the model output file may be 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. You may have a postprocessing program which interpolates the model-generated stress array 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

```
prestres
stres3d
postres
```

Here PRESTRES and POSTRES are the model pre- and postprocessors respectively; STRES3D is the stress model itself.

You can get even more complicated than this if you wish. For example, a problem that can arises 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 ground water model, MODFLOW, requires a variable HCLOSE which determines the precision with which heads are calculated by its preconditioned conjugate gradient matrix solution package. As discussed in Section 2.3.3, 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 conjugate gradient method may never converge to a point where the maximum head correction between successive conjugate gradient iterations is less than HCLOSE, roundoff errors causing slight oscillatory behaviour. In this case MODFLOW will terminate execution with an error message. Unfortunately, it may be very difficult to predict when this will occur; behaviour of the solution method may be perfect for one parameter set 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.

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. (See a DOS manual or help file for a description of this setting; most compilers allow you to set the *errorlevel* 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 Example 4.3.

```
@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
```

Example 4.3 A batch file called by PEST as the model.

(Note that there are alternative, simpler solutions to the MODFLOW convergence problem discussed here; the purpose of this example is to demonstrate the type of batch processing that may be useful as a PEST model run.)

The variations on the content of a model batch file 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 the models or just from the file(s) generated by the last. Another possibility is that the model batch file may call the same model 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. Furthermore, as shown in the example above, you can insert intelligence into the way component models are run through the use of the *errorlevel* variable.

4.2.9 Model Input/Output

In this section of the PEST control file you must relate PEST template files to model input files and PEST instruction files to model output files. You will already have informed PEST of the respective numbers of these files through the PEST control variables NTPLFLE and NINSFLE. See Example 4.1 for the structure of the "model input/output" section of the PEST control file.

For each model input file - PEST template file pair there should be a line within the "model input/output" section of the PEST control file containing two entries, viz. 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 3 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 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 explained in Chapter 3, 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 3 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.

4.2.10 Prior Information

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 is written to this section of the PEST control file in a manner not unlike the way in which you would write it down on paper yourself; however certain strict protocols must be observed. Refer to Example 4.2 for an instance of a PEST control file containing prior information.

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 article to several successive lines.

Prior information lines must adhere to the syntax set out in Example 4.1. The protocol is repeated here for ease of reference.

```
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of the NPRIOR articles of prior information)
```

Example 4.4 The syntax of a prior information line.

Each prior information article must begin with a prior information label (the character variable PILBL in Example 4.4). Like all other names used by PEST, this label must be no more than twelve 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; ie. 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, viz. 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 optimisation 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 with 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 article 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 corresponding measurements.

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 article 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 viz. PIVAL, WEIGHT and OBGNME. The first of these is the value of the right side of the prior information equation. The second is the weight assigned to the article of prior information in the parameter estimation process. As for observation weights, the prior information weight should ideally be inversely proportional to the standard deviation of the prior information value (PIVAL); it can be zero if you wish but must not be negative. in practice the weights should be chosen such that the prior information equation neither dominates the objective function or is dwarfed by other components of the objective function. In choosing observation and prior information weights, remember that the weight is multiplied by its respective residual and then squared before being assimilated into the objective function.

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 carrying out the parameter estimation 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. If desired, a particular item of prior information can belong to the same group as an observation cited in the "observation data" section of the PEST control file. However it is difficult to see why this would be done because, under normal circumstances, the user will want to know the relative contributions made to the objective function by observations and prior information separately.

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 prior information article 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
```

4.3 Observation Covariances

4.3.1 Using an Observation Covariance Matrix Instead of Weights

As was discussed in Section 2.1.2, the use of observation weights in calculating the objective function is based on the premise that observations are independent, ie. 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 assume statistical independence of these observation types. 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. See Section 2.1.11 for more details.

Use of an observation covariance matrix can be particularly useful when 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). In this case it may be a good idea to include that correlation in the inversion process by assigning a covariance matrix to the prior information equations that reflects the observed parameter interdependence, rather than a set of weights based on the false premise that the value of each parameter is independent of that of its neighbours.

Observation correlation may be important in other situations as well. For example consider the case where, for a particular ground water model, the extent of outflow from the ground water domain into two neighbouring reaches of a stream is used in the model calibration process. Consider also (as often occurs in practice) that the total outflow into both of the neighbouring reaches can be more accurately measured than the outflow into each individual reach. However it may be considered desirable for a particular model application that the model be calibrated using both of the individual reach outflows 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, rather than to ignore this interdependence through the assignment of separate, individual weights, when undertaking the inversion process.

4.3.2 Supplying the Observation Covariance Matrix to PEST

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 for a group of observations.

More than one covariance matrix can be supplied to PEST for use in the parameter estimation 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. Example 4.5 shows a simple PEST control file in which two covariance matrices are supplied, one for the observation group "obsgp1" and the other for the observation group "obsgp2", the latter being used for prior information.

```
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
```

Example 4.5. A PEST control file citing two covariance matrices.

The following rules must be obeyed when using one or more observation covariance matrices in a PEST run.

1. The name of a text file containing an observation covariance matrix (or rather, a matrix related to an observation covariance matrix by an unknown constant of proportionality), can be provided in the PEST control file following the name of the observation group to which the matrix pertains in the "observation groups" section of the PEST control file; in Example 4.5 the names of these observation covariance matrix files are *cov1.dat* and *cov2.dat*.

2. A covariance matrix file must contain a square symmetric matrix of dimension *n,* where *n* is the number of observations belonging to the observation group to which the covariance matrix pertains. Thus every observation belonging to the pertinent observation group must be involved in the covariance matrix. Example 4.6 illustrates a covariance matrix file.

3. All diagonal elements of the covariance matrix must be positive. While the matrix should, theoretically, be positive definite to qualify as a covariance matrix, a symmetric matrix will be acceptable. However the matrix must be such that it is possible to calculate eigenvectors and eigenvalues for that matrix without incurring numerical difficulties (this will rarely be a problem).

4. Elements of the covariance matrix, as represented in the covariance matrix file, must be space or comma-delimited. A line of this matrix can wrap around to the next line if it is too long. However each row of the matrix must begin on a new line.

5. Whether or not a covariance matrix is supplied for a particular observation group, weights must still be supplied for members of that group in the "observation data" section of the PEST control file. However these weights will be ignored by PEST (including a weight of zero that may be assigned to a certain observation in order to "take it out" of the parameter estimation process).

6. Observation groups used for prior information and those used for actual observations must be separate when one or more covariance matrices are supplied for use in the inversion process. Thus a particular observation group cannot have members which are both observations and prior information equations.

```
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
```

Example 4.6 Example of a covariance matrix file.

At the end of the inversion process the true covariance matrices pertaining to various observation groups can be calculated from user-supplied covariance matrices through multiplication by the reference variance determined through the parameter estimation process (ie. σ^2 of equation 2.5). Recall from Section 2.1.2 that variances and covariances represented in covariance matrices supplied to PEST by the user will be related to true observation variances and covariances by a constant of proportionality that is unknown before completion of the inversion process. For the sake of consistency with observations for which a covariance matrix is not supplied, this constant of proportionality should be the same as that by which observation variances are related to the inverse square of observation weights. In practice, however, the user should simply ensure that, as always, weights and covariance matrices are such that no observation group either dominates the parameter estimation process or is dominated by other observation groups.

4.3.3 PEST Outputs

When one or more observation covariance matrices are supplied to PEST as part of its input dataset, PEST's output dataset is a little different from that which is recorded if no covariance matrices are supplied. While PEST outputs are treated in detail in the next chapter, these differences are now briefly outlined.

4.3.3.1 Echoing of Covariance Matrices

Before undertaking the parameter estimation process, PEST records much of the information that it reads from the PEST control file to its run record file. This information includes the contents of any covariance matrix files that are supplied to it in its input dataset.

When echoing observation weights to its run record file, the weights supplied for observations belonging to an observation group for which a covariance matrix has been supplied are not recorded, for these weights are not used in the inversion process. Rather, the character string "Cov. Mat." is recorded in place of the pertinent weights to remind the user that an observation covariance matrix is used in their stead.

4.3.3.2 Objective Function

Calculation of the objective function, and of the contribution to the objective function made by various observation groups, takes account of the fact that a covariance matrix is supplied for at least one group of observations.

4.3.3.3 Residuals

Measurements, together with their model-generated counterparts calculated on the basis of best-fit parameters, are tabulated at the end of the run record file; observation weights are also tabulated with this data. For those observations which are associated with a covariance matrix, the character string "Cov. Mat." replaces the observation weight in this table, this indicating, once again, that the latter are ignored in all calculations pertaining to these observations undertaken by PEST.

At the end of the parameter estimation process PEST records measurements, their model-generated counterparts, residuals, observation weights, and a number of functions of these in a "residuals file". The format of this file is such that the data contained therein is suitable for importation into a spreadsheet for further mathematical analysis. Where tabulated functions of those observations for which a covariance matrix is supplied involve observation weights, these functions are not calculated and recorded by PEST, for the weights supplied by the user for these observations are not used in the inversion process. Instead the "Cov. Mat." string is written in place of the redundant observation weight, and "na" (for "not applicable") is recorded in place of any functions which depend on these weights.

If at least one observation covariance matrix is supplied in its input dataset, PEST records an additional residuals file called a "rotated residuals file". This has the same filename base as the ordinary residuals file (ie. the filename base of the PEST control file), but is given an extension of ".rsr". Whereas the normal residuals file tabulates measurements, their model-generated counterparts, the residuals calculated therefrom, and various functions of these quantities (see Section 5.3.4), the rotated residuals file tabulates "rotated measurements", their rotated model-generated counterparts, residuals calculated therefrom, and the same functions of these quantities. Because, through the use of rotated observations the observation covariance matrix is diagonalised, weights can be used in the calculation of these various functions. As is explained in Section 2.1.11, these weights are actually the reciprocals of the square roots of the eigenvalues of the original observation covariance matrix supplied by the user.

Where a covariance matrix is supplied for only a few of the many observations used in the parameter estimation process, most of the entries in the rotated residuals file will be the same as those found in the normal residuals file. However entries pertaining to observation groups for which a covariance matrix is supplied will be different. 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. Rotated observations are listed in order of decreasing observation weight (ie. in order of increasing eigenvalues of the original covariance matrix). New names for these observations are formulated in the order in which these observations are supplied in the original PEST control file. It is important to note that a rotated observation whose name is formulated by adding the string "_r" to the name of the 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.

4.3.3.4 Analysis of Residuals

PEST calculates and records a number of basic statistics pertaining to optimised residuals to the end of its run record file. Due to the fact that these statistics are calculated on the basis of weighted residuals, rather than the residuals themselves, PEST calculates them using *rotated* residuals rather than true residuals for those observation groups for which a covariance matrix is supplied. The fact that rotated residuals, rather than direct residuals, are used in this calculation is recorded on the run record file. Also, where the names of any such rotated residuals are cited, the "_r" suffix appended to a residual's name indicates its rotated status.

5. Running PEST

5.1 How to Run PEST

5.1.1 Checking PEST's Input Data

PEST's input file requirements have been discussed in detail in the previous two chapters. Before submitting these files to PEST for a parameter estimation run, you should check that all information contained in them is syntactically correct and consistent. This can be done using the utility programs PESTCHEK, TEMPCHEK and INSCHEK described in Chapter 10 of this manual.

PEST carries out some checking of its input dataset itself; if there are any syntax errors in any of these 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, as the coding required to achieve this would take up too much memory, this memory being reserved for array storage for PEST and, possibly, the model. 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 optimisation 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.

5.1.2 Versions of PEST

As explained in Chapter 1, there are two versions of PEST. Each can be run by typing the name of the pertinent executable file at the command prompt.

PEST

The "single window" version of PEST (contained in the *pest.exe* executable program) is the simpler version of PEST to use. In this version of PEST, the model and PEST share the same window. Hence screen output from one will cause screen output from the other to scroll away out of sight.

The single window version of PEST is run using the command

```
pest case [/r] [/j]
```

where *case* is the filename base of the PEST control file (PEST automatically adds the extension ".pst") and "/r" or "/j" is an optional restart switch.

PPEST

PPEST is Parallel PEST, the operation of which is fully described in Chapter 9. Parallel

PEST is contained in the *ppest.exe* executable. When a user runs Parallel PEST, he/she must also run one or a number of "slaves" which, in turn, run the model. These slaves can reside on the same machine as Parallel PEST, or on other machine(s) with which the PEST machine is networked. Because model runs can be undertaken simultaneously on different machines during calculation of the Jacobian matrix, the savings in overall optimisation time through the use of Parallel PEST can be considerable. It should be noted, however, that due to the overheads involved in communicating with one or a number of slaves, parameter estimation for a small model with a short run time may actually be larger when using Parallel PEST than when using the single window version of PEST. The considerable efficiencies involved in parallelisation of the parameter estimation process are only fully realised where model run-times are of the order of 30 seconds or greater.

Parallel PEST is a little more complex to run than the single window version of PEST because an extra PEST input file (called the "run management file") must be prepared. Also, as well as starting PEST, the user must also start each of the slaves. However it is more than worth the extra trouble where model run times are large and adjustable parameters are many.

While Parallel PEST was built for the purpose of running a model simultaneously on a number of different machines across a network, it can also be used to run a single instance of the model on a single machine. Doing this has the advantage that the model and PEST operate in different windows; hence the screen output of one does not interfere with the screen output of the other.

Parallel PEST is run using the command

```
ppest case [/r] [/j]
```

where *case* is the filename base of the PEST control file (PPEST automatically adds the extension ".pst") and "/r" and "/j" are optional restart switches.

For more information on running Parallel PEST, see Chapter 9 and the "Frequently Asked Questions" in Chapter 13.

5.2 The PEST Run Record

5.2.1 An Example

As PEST executes, it writes a detailed record of the parameter estimation process to file *case.rec*, where *case* is the filename base of the PEST control file to which it is directed through the PEST command line. Example 5.1 shows such a run record file; the PEST control file corresponding to Example 5.1 is that shown in Example 4.2. Note that this example does not demonstrate a very good fit between measurements and 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 parameter estimation mode in order to produce the run record demonstrated in Example 5.1. As will be discussed in Sections 6 and 7, the run record produced as an outcome of a PEST run in predictive analysis or regularisation modes is slightly different.

Example 5.1 A PEST run record file; Example 4.2 shows the corresponding PEST control file.

```
                      PEST RUN RECORD: CASE  manual


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:-

ves


Model interface files:-

   Templates:
      ves.tpl
   for model input files:
      ves.inp

   (Parameter values written using single precision protocol.)
   (Decimal point always included.)

   Instruction files:
      ves.ins
   for reading model output files:
      ves.out


Derivatives calculation:-

Param Increment   Increment   Increment   Forward or    Multiplier  Method
group type                  low bound   central       (central)   (central)
 ro   relative    1.0000E-03  1.0000E-05   switch         2.000     parabolic
 h    relative    1.0000E-03  1.0000E-05   switch         2.000     parabolic


Parameter definitions:-

Name  Trans-       Change       Initial        Lower          Upper       Group
      formation    limit        value          bound          bound
ro1   fixed          na       0.500000          na             na         none
ro2   log          factor      5.00000       0.100000        10.0000      ro
ro3   tied to ro2    na       0.500000          na             na         ro
h1    none         factor      2.00000       5.000000E-02    100.000      h
h2    log          factor      5.00000       5.000000E-02    100.000      h

Name      Scale          Offset
ro1       1.00000       0.000000
ro2       1.00000       0.000000
ro3       1.00000       0.000000
h1        1.00000       0.000000
h2        1.00000       0.000000


Prior information:-

Prior info   Factor             Parameter          Prior         Weight
name                                               information
  pi1        1.00000       *        h1       =      2.00000        3.000
  pi2        1.00000       *    log[ro2]     +
             1.00000       *    log[h2]      =      2.60260        2.000

Prior Info Name   Observation Group
  pi1               group_4
  pi2               group_4


Observations:-

Observation name    Observation       Weight   Group
     ar1             1.21038          1.000    group_1
     ar2             1.51208          1.000    group_1
     ar3             2.07204          1.000    group_1
```

```
     ar4             2.94056          1.000    group_1
     ar5             4.15787          1.000    group_1
     ar6             5.77620          1.000    group_1
     ar7             7.78940          1.000    group_2
     ar8             9.99743          1.000    group_2
     ar9             11.8307          1.000    group_2
     ar10            12.3194          1.000    group_2
     ar11            10.6003          1.000    group_2
     ar12            7.00419          1.000    group_2
     ar13            3.44391          1.000    group_2
     ar14            1.58279          1.000    group_2
     ar15            1.10380          1.000    group_3
     ar16            1.03086          1.000    group_3
     ar17            1.01318          1.000    group_3
     ar18            1.00593          1.000    group_3
     ar19            1.00272          1.000    group_3



Inversion control settings:-

   Initial lambda                                                 :  5.0000
   Lambda adjustment factor                                       :  2.0000
   Sufficient new/old phi ratio per iteration                     : 0.40000
   Limiting relative phi reduction between lambdas                : 3.00000E-02
   Maximum trial lambdas per iteration                            :  10

   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

   Relative phi reduction below which to begin use of
   central derivatives                                            : 0.10000

   Relative phi reduction indicating convergence                  : 0.10000E-01
   Number of phi values required within this range                :   3
   Maximum number of consecutive failures to lower phi            :   3
   Maximum relative parameter change indicating convergence       : 0.10000E-01
   Number of consecutive iterations with minimal param change     :   3
   Maximum number of optimisation iterations                      :  30


                                OPTIMISATION RECORD


INITIAL CONDITIONS:
Sum of squared weighted residuals (ie phi) =  523.8
Contribution to phi from observation group "group_1" = 127.3
Contribution to phi from observation group "group_2" = 117.0
Contribution to phi from observation group "group_3" = 185.2
Contribution to phi from observation group "group_4" = 94.28


      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:  523.8
   Contribution to phi from observation group "group_1": 127.3
   Contribution to phi from observation group "group_2": 117.0
   Contribution to phi from observation group "group_3": 185.2
   Contribution to phi from observation group "group_4": 94.28

   Lambda =  5.000     ----->
      phi =  361.4      (  0.69 of starting phi)

   Lambda =  2.500     ----->
      phi =  357.3      (  0.68 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  357.3

      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          1.94781                       h1          2.00000
        h2          10.4413                       h2          5.00000
   Maximum   factor parameter change:  2.088  [h2]
   Maximum relative parameter change:  1.088  [h2]


OPTIMISATION ITERATION NO.        :    2
   Model calls so far             :    6
   Starting phi for this iteration:  357.3
   Contribution to phi from observation group "group_1": 77.92
   Contribution to phi from observation group "group_2": 103.8
   Contribution to phi from observation group "group_3": 121.3
   Contribution to phi from observation group "group_4": 54.28

   Lambda =  1.250     ----->
      parameter "ro2" frozen: gradient and update vectors out of bounds
      phi =  252.0      (  0.71 of starting phi)

   Lambda = 0.6250     ----->
      phi =  243.6      (  0.68 of starting phi)

   Lambda = 0.3125     ----->
      phi =  235.9      (  0.66 of starting phi)

   Lambda = 0.1563     ----->
      phi =  230.1      (  0.64 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  230.1

      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          1.41629                     h1          1.94781
        h2          31.3239                     h2          10.4413
   Maximum   factor parameter change:  3.000  [h2]
   Maximum relative parameter change:  2.000  [h2]


OPTIMISATION ITERATION NO.        :    3
   Model calls so far             :   13
   Starting phi for this iteration:  230.1
   Contribution to phi from observation group "group_1": 29.54
   Contribution to phi from observation group "group_2": 84.81
   Contribution to phi from observation group "group_3": 91.57
   Contribution to phi from observation group "group_4": 24.17

   All frozen parameters freed

   Lambda = 7.8125E-02 ----->
      parameter "ro2" frozen: gradient and update vectors out of bounds
      phi =  89.49      (  0.39 of starting phi)

   No more lambdas: phi is now less than 0.4000 of starting phi
   Lowest phi this iteration:  89.49

      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         0.472096                     h1          1.41629
        h2          34.3039                     h2          31.3239
   Maximum   factor parameter change:  3.000  [h1]
   Maximum relative parameter change: 0.6667  [h1]


OPTIMISATION ITERATION NO.        :    4
   Model calls so far             :   17
   Starting phi for this iteration:  89.49
   Contribution to phi from observation group "group_1": 9.345
   Contribution to phi from observation group "group_2": 34.88
   Contribution to phi from observation group "group_3": 21.57
   Contribution to phi from observation group "group_4": 23.69

   All frozen parameters freed

   Lambda = 3.9063E-02 ----->
      parameter "ro2" frozen: gradient and update vectors out of bounds
      phi =  79.20      (  0.89 of starting phi)

   Lambda = 1.9531E-02 ----->
      phi =  79.19      (  0.88 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
```

```
   Lowest phi this iteration:  79.19

      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         0.157365                     h1         0.472096
        h2          44.2189                     h2          34.3039
   Maximum   factor parameter change:  3.000  [h1]
   Maximum relative parameter change: 0.6667  [h1]


OPTIMISATION ITERATION NO.        :    5
   Model calls so far             :   22
   Starting phi for this iteration:  79.19
   Contribution to phi from observation group "group_1": 6.920
   Contribution to phi from observation group "group_2": 22.45
   Contribution to phi from observation group "group_3": 14.88
   Contribution to phi from observation group "group_4": 34.94

   All frozen parameters freed

   Lambda = 9.7656E-03 ----->
      parameter "ro2" frozen: gradient and update vectors out of bounds
      phi =  64.09      (  0.81 of starting phi)

   Lambda = 4.8828E-03 ----->
      phi =  64.09      (  0.81 of starting phi)

   Lambda = 1.9531E-02 ----->
      phi =  64.09      (  0.81 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  64.09

      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         0.238277                     h1         0.157365
        h2          42.4176                     h2          44.2189
   Maximum   factor parameter change:  1.514  [h1]
   Maximum relative parameter change: 0.5142  [h1]


OPTIMISATION ITERATION NO.        :    6
   Model calls so far             :   28
   Starting phi for this iteration:  64.09
   Contribution to phi from observation group "group_1": 6.740
   Contribution to phi from observation group "group_2": 18.98
   Contribution to phi from observation group "group_3": 10.53
   Contribution to phi from observation group "group_4": 27.84

   All frozen parameters freed

   Lambda = 1.9531E-02 ----->
      parameter "ro2" frozen: gradient and update vectors out of bounds
      phi =  63.61      (  0.99 of starting phi)

   Lambda = 9.7656E-03 ----->
      phi =  63.61      (  0.99 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  63.61
   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         0.265320                     h1         0.238277
        h2          42.2249                     h2          42.4176
   Maximum   factor parameter change:  1.113  [h1]
   Maximum relative parameter change: 0.1135  ]h1]


OPTIMISATION ITERATION NO.        :    7
   Model calls so far             :   33
   Starting phi for this iteration:  63.61
   Contribution to phi from observation group "group_1": 3.679
   Contribution to phi from observation group "group_2": 32.58
   Contribution to phi from observation group "group_3": 0.111
   Contribution to phi from observation group "group_4": 27.24
```

```
   All frozen parameters freed

   Lambda = 4.8828E-03 ----->
      parameter "ro2" frozen: gradient and update vectors out of bounds
      phi =  63.59      (  1.00 of starting phi)

   Lambda = 2.4414E-03 ----->
      phi =  63.59      (  1.00 of starting phi)

   Lambda = 9.7656E-03 ----->
      phi =  63.59      (  1.00 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  63.59

      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         0.261177                     h1         0.265320
        h2          42.2006                     h2          42.2249
   Maximum   factor parameter change:  1.016     [h1]
   Maximum relative parameter change: 1.5615E-02 [h1]

   Optimisation complete: the  3 lowest phi's are within a relative distance
                          of eachother of 1.000E-02
   Total model calls:   42


                           OPTIMISATION RESULTS


Adjustable parameters ----->

Parameter      Estimated           95% percent confidence limits
               value               lower limit       upper limit
 ro2           10.0000            0.665815          150.192
 h1           0.261177            -1.00256          1.52491
 h2            42.2006            0.467914          3806.02

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


Observations ----->

Observation    Measured        Calculated      Residual        Weight    Group
               value           value
 ar1           1.21038         1.64016       -0.429780         1.000    group_1
 ar2           1.51208         2.25542       -0.743340         1.000    group_1
 ar3           2.07204         3.03643       -0.964390         1.000    group_1
 ar4           2.94056         3.97943        -1.03887         1.000    group_1
 ar5           4.15787         5.04850       -0.890630         1.000    group_1
 ar6           5.77620         6.16891       -0.392710         1.000    group_1
 ar7           7.78940         7.23394        0.555460         1.000    group_2
 ar8           9.99743         8.12489         1.87254         1.000    group_2
 ar9           11.8307         8.72551         3.10519         1.000    group_2
 ar10          12.3194         8.89590         3.42350         1.000    group_2
 ar11          10.6003         8.40251         2.19779         1.000    group_2
 ar12          7.00419         6.96319       4.100000E-02      1.000    group_2
 ar13          3.44391         4.70412        -1.26021         1.000    group_2
 ar14          1.58279         2.56707       -0.984280         1.000    group_2
 ar15          1.10380         1.42910       -0.325300         1.000    group_3
 ar16          1.03086         1.10197       -7.111000E-02     1.000    group_3
 ar17          1.01318         1.03488       -2.170000E-02     1.000    group_3
 ar18          1.00593         1.01498       -9.050000E-03     1.000    group_3
 ar19          1.00272         1.00674       -4.020000E-03     1.000    group_3


Prior information ----->

Prior          Provided        Calculated      Residual        Weight    Group
```

```
information    value          value
 pi1           2.00000        0.261177         1.73882         3.000    group_4
 pi2           2.60260         2.62532       -2.271874E-02     2.000    group_4

See file TEMP3.RES for more details of residuals in graph-ready format.
See file TEMP3.SEO for composite observation sensitivities.

Objective Function ----->

  Sum of squared weighted residuals (ie phi)                =   63.59
  Contribution to phi from observation group "group_1"      =   3.686
  Contribution to phi from observation group "group_2"      =   32.58
  Contribution to phi from observation group "group_3"      =  0.1115
  Contribution to phi from observation group "group_4"      =   27.21


Correlation Coefficient ----->
  Correlation coefficient                                   =  0.9086


Analysis of residuals ----->

  All residuals:-
     Number of residuals with non-zero weight               =    21
     Mean value of non-zero weighted residuals              = -0.4399
     Maximum weighted residual [observation "ar13"]         =   1.260
     Minimum weighted residual [observation "pi1"]          =  -5.216
     Standard variance of weighted residuals                =   3.533
     Standard error of weighted residuals                   =   1.880

     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.

  Residuals for observation group "group_1":-
     Number of residuals with non-zero weight               =     6
     Mean value of non-zero weighted residuals              =  0.7424
     Maximum weighted residual [observation "ar4"]          =   1.038
     Minimum weighted residual [observation "ar6"]          =  0.3916
     "Variance" of weighted residuals                       =  0.6144
     "Standard error" of weighted residuals                 =  0.7838

     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 "group_2":-
     Number of residuals with non-zero weight               =     8
     Mean value of non-zero weighted residuals              =  -1.119
     Maximum weighted residual [observation "ar13"]         =   1.260
     Minimum weighted residual [observation "ar10"]         =  -3.424
     "Variance" of weighted residuals                       =   4.072
     "Standard error" of weighted residuals                 =   2.018

     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 "group_3":-
     Number of residuals with non-zero weight               =     5
     Mean value of non-zero weighted residuals              =  8.6256E-02
     Maximum weighted residual [observation "ar15"]         =  0.3254
     Minimum weighted residual [observation "ar19"]         =  4.0200E-03
     "Variance" of weighted residuals                       =  2.2300E-02
     "Standard error" of weighted residuals                 =  0.1493

     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 "group_4":-
     Number of residuals with non-zero weight               =     2
     Mean value of non-zero weighted residuals              =  -2.585
     Maximum weighted residual [observation "pi2"]          =  4.5451E-02
     Minimum weighted residual [observation "pi1"]          =  -5.216
     "Variance" of weighted residuals                       =   13.61
     "Standard error" of weighted residuals                 =   3.689

     Note: the above "variance" was obtained by dividing the sum of squared
     residuals by the number of items with non-zero weight.


Covariance Matrix ----->
```

```
          ro2           h1            h2
ro2      0.3136       4.8700E-03  -0.4563
h1       4.8700E-03   0.3618       1.3340E-02
h2      -0.4563       1.3340E-02   0.8660


Correlation Coefficient Matrix ----->

          ro2           h1            h2
ro2      1.000        1.4457E-02  -0.8756
h1       1.4457E-02    1.000       2.3832E-02
h2      -0.8756       2.3832E-02    1.000


Normalized eigenvectors of covariance matrix ----->
         Vector_1      Vector_2      Vector_3
ro2     -0.8704       -3.6691E-02  -0.4909
h1       3.5287E-02   -0.9993       1.2121E-02
h2      -0.4910       -6.7718E-03   0.8711


Eigenvalues ----->

         5.6045E-02    0.3621        1.123
```

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 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 Example 5.1, "na" is used a number of times to indicate that a particular PEST input variable has no effect on the optimisation 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 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 with which they are cited in the PEST control file. The fact that PEST adjusts its internal representations of parameter values such that they are expressed with the same degree of precision as that with which they are written to the model input files has already been discussed (see Section 3.2). For consistency, PEST's internal representation of parameter bounds is adjusted in the same way.

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 estimation process in earnest, beginning with the first optimisation 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 optimisation iteration.

During the first iteration of Example 5.1, PEST tests two Marquardt lambdas; because the second lambda results in an objective function fall of less than 0.03 (ie. PHIREDLAM)

relative to the first one tested, PEST does not test any further lambdas. Instead it progresses to the next optimisation iteration after listing both the updated parameter values as well as those from which the updated parameter set was calculated, viz. those at the commencement of the optimisation iteration. Note that the only occasion on which the "previous parameter values" recorded at the end of an optimisation iteration do not correspond with those determined during the previous optimisation iteration is when the switch to three-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 optimisation iteration the parameter set resulting in the lowest objective function value achieved so far.

At the end of each optimisation iteration PEST records either two or three (depending on the input settings) very important pieces of information; in the case of Example 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 either factor-limited or relative-limited; in Example 5.1 all adjustable parameters are factor-limited with a factor limit of 3.0. A suitable setting for the factor and relative change limits (ie. FACPARMAX and RELPARMAX) may be crucial in achieving optimisation stability. Note that, along with the value of the maximum factor or parameter change encountered during the optimisation iteration, PEST also records the name of the parameter that underwent this change. This information may be crucial in deciding which, if any, parameters should be temporarily held at their current values should trouble be encountered in the optimisation process. For details of the options for user-intervention, see Section 5.6 of this manual.

The recording of the maximum factor and relative parameter changes at the end of each iteration allows you to judge whether you have set these vital variables (ie. FACPARMAX and RELPARMAX) wisely. In the present case only the maximum factor change is needed because no parameters are relative-limited; the maximum relative parameter change is recorded, however, because one of the termination criteria involves the use of relative parameter changes. Note that had some of the parameters in Example 5.1 been relative-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 and the maximum relative parameter change would have been provided for relative-limited parameters. 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 Example 5.1 shows that in iteration 2, one of the parameters, viz. "h2", incurs the maximum allowed factor change, thus limiting the magnitude of the parameter upgrade vector. In optimisation iterations 3 and 4, parameter "h1" limits the magnitude of the parameter upgrade vector through incurring the maximum allowed parameter factor change. It is possible that convergence for this case would have been achieved much faster if FACPARMAX on the PEST control file were set higher than 3.0.

At the beginning of the second optimisation iteration, parameter "ro2" is at its upper bound. After calculating the Jacobian matrix and formulating and solving equation 2.23, 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. The two-step process by which PEST judges whether to freeze a parameter which is at its upper or lower limit is explained in Section 2.2.3. Note that at the beginning of optimisation iteration 3, parameter "ro2" is released again in case, with the upgrading of the other adjustable parameters during the previous optimisation iteration, it wants to move back into the internal part of its domain.

In the third optimisation iteration only a single Marquardt lambda is tested, the objective function having been lowered to below 0.4 times its starting value for that iteration through the use of this single lambda; 0.4 is the user-supplied value for the PEST control variable PHIRATSUF.

During the fifth optimisation iteration three lambdas are tested. The second results in a raising of the objective function over the first (though this is not apparent in the run record because "phi", the objective function, is not written with sufficient precision to show it), so PEST tests a lambda which is higher than the first. For the case illustrated in Example 5.1, when lambda is raised or lowered it is adjusted using a factor of 2.0, this being the user-supplied value for the PEST control variable RLAMFAC. For optimisation iteration 6, the first lambda tested is the same as the most successful one for the previous iteration, viz. 1.9531E-02. However, for each of the previous iterations, where the objective function was improved through lowering lambda during the iteration prior to that, the starting lambda is lower by a factor of 2.0 (ie. RLAMFAC) than the most successful lambda of the previous iteration.

At the end of optimisation iteration 6 PEST calculates that the relative reduction in the objective function from that achieved in iteration 5 is less that 0.1; ie. 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 those groups from now on. Note that in Example 5.1, 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. However 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 optimisation process moving again.

The optimisation process of Example 5.1 is terminated at the end of optimisation iteration 7, after the lowest 3 (ie. NPHISTP) objective function values are within a relative distance of 0.01 (ie. PHIREDSTP) of each other.

Note that where PEST lists the current objective function value at the start of the optimisation process and at the start of each optimisation iteration, it also lists the contribution made to the objective function by each observation group (including the observation group "group_4" 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 optimisation process again.

5.2.4 Optimised Parameter Values and Confidence Intervals

After completing the parameter estimation process, PEST prints the outcomes of this process to the third section of the run record file. First it lists the optimised parameter values. It does this in three stages; the adjustable parameters, then the tied parameters and, finally, any fixed parameters. PEST calculates 95% confidence limits for the adjustable parameters. However, you should note carefully the following points about confidence limits.

- Confidence limits can only be obtained if the covariance matrix has been calculated. If, for any reason, it has not been calculated (eg. because $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ of equation 2.17 could not be inverted) confidence limits will not be provided.

- As noted in the PEST run record itself, parameter confidence limits are calculated on the basis of the same linearity assumption which was used to derive the equations for parameter improvement implemented in each PEST optimisation 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. This is readily apparent in Example 5.1.

- No account is taken of parameter upper and lower bounds in the calculation of 95% confidence intervals. Thus an upper or lower confidence limit can lie well outside a parameter's allowed domain. In Example 5.1, the upper confidence limits for both "ro2" and "h2" lie well above the allowed upper bounds for these parameters, as provided by the parameter input variable PARUBND for each of these parameters; similarly the lower confidence limit for parameter "h1" lies below its lower bound (PARLBND) of 0.05. PEST does not truncate the confidence intervals at the parameter domain boundaries so as not to provide an unduly optimistic impression of parameter certainty.

- The parameter confidence intervals are highly dependent on the assumptions underpinning the model. If the model has too few parameters to accurately simulate a particular system, the optimised objective function will be large and then so too, through equations 2.5 and 2.17, will be the parameter covariances and, with them, the parameter confidence intervals. However, if a model has too many parameters, the objective function may well be small, but some parameters may be highly correlated with each other due to an inability on the part of a possibly limited measurement set to uniquely determine each parameter of such a complex model; this will give rise to large covariance values (and hence large confidence intervals) for the correlated parameters.

Notwithstanding the above limitations, the presentation of 95% confidence limits provides a useful means of comparing the certainty with which different parameter values are estimated by PEST. In Example 5.1 it is obvious that parameters "ro2" and "h2" (particularly "h2") are estimated with a large margin of uncertainty. This is because these two parameters are well

correlated; this means that they can be varied in harmony and, provided one is varied in a manner that properly complements the variation of the other, there will be little effect on the objective function. Hence while the objective function may be individually sensitive to each one of these parameters, it appears to be relatively insensitive to both of them if they are varied in concert. This illustrates the great superiority of using covariance and eigenvector analysis over the often-used "sensitivity analysis" method of determining parameter reliability.

Confidence limits are not provided for tied parameters. The parent parameters of all tied parameters are estimated with the tied parameters "riding on their back"; hence the confidence intervals for the respective parent parameters reflect their linkages to the tied parameters.

Note that at the end of a PEST optimisation run a listing of the optimised parameter values can also be found in the PEST parameter value file *case.par*.

5.2.5 Observations and Prior Information

After it has written the optimised parameter set to the run record file, PEST records the measured observation values, together with their model-generated counterparts calculated on the basis of the optimised parameter set. The differences between the two (ie. the residuals) are also listed, together with the user-supplied set of observation weights. Following the observations, the user-supplied and model-optimised prior information values are listed; a prior information value is the number on the right side of the prior information equation. As for the observations, residuals and user-supplied weights are also tabulated.

Tabulated residuals and weighted residuals can also be found in file *case.res*; see Section 5.3.4. Composite observation sensitivities can be found in file *case.seo*; see Section 5.3.3.

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 2.43, is next listed.

5.2.8 Analysis of Residuals

The next section of the run record file lists a number of statistics pertaining to observation 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 The Parameter Covariance Matrix

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 which is neither fixed nor tied. The order in which the rows (and columns) are arranged is the same as the order of occurrence of the adjustable parameters in the previous listing of the optimised 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 parameter estimation process (see Chapter 2), the elements of the 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 pertaining to 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 in so far as the linearity assumption, upon which their calculation is based, is valid.

The diagonal elements of the covariance matrix are the variances of the adjustable parameters; for Example 5.1 the 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. With log("h2") having a variance of 0.866 (and hence a standard deviation of 0.931), and bearing in mind that the number "1" in the log domain represents a factor of 10 in untransformed parameter space, it is not hard to see why the 95% confidence interval cited for parameter "h2" is so wide.

The off-diagonal elements of the covariance matrix represent the covariances between parameter pairs; thus, for example, the element in 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; ie. after eight numbers have been written, PEST will start a new line to write the ninth number. Similarly if there are more than sixteen adjustable parameters, the seventeenth number will begin on a new line. Note, however, that every new row of the covariance matrix begins on a new line of the run record file.

5.2.10 The Correlation Coefficient Matrix

The correlation coefficient matrix is calculated from the covariance matrix through equation 2.7. The correlation coefficient matrix has the same number of rows and columns as the covariance matrix; furthermore the manner in which these rows and columns are related to adjustable parameters (or their logs) is identical to that for the covariance matrix. Like the covariance matrix, the correlation coefficient matrix is symmetric.

The diagonal elements of the correlation coefficient matrix are always unity; the off-diagonal elements are always between 1 and -1. The closer that an off-diagonal element is to 1 or -1, the more highly correlated are the parameters corresponding to the row and column numbers of that element. Thus, for the correlation coefficient matrix of Example 5.1, the logs of

parameters "ro2" and "h2" show medium to high correlation, as is indicated by the value of elements (1,3) and (3,1) of the correlation coefficient matrix, viz. -0.8756. This explains why, individually, these parameters are determined with a high degree of uncertainty in the parameter estimation process, as evinced by their wide confidence intervals.

5.2.11 The Normalised Eigenvector Matrix and the Eigenvalues

The eigenvector matrix is composed of as many columns as there are adjustable parameters, each column containing a normalised eigenvector. Because the covariance matrix is positive definite, these eigenvectors are real and orthogonal; they represent the directions of the axes of the probability "ellipsoid" in the *n*-dimensional space occupied by the *n* 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 semiaxis of the probability ellipsoid in *n*-dimensional adjustable parameter space.

If the ratio of a particular eigenvalue to the lowest eigenvalue pertaining to the parameter estimation problem is particularly large, then the respective eigenvector defines a direction of relative insensitivity in parameter space. The eigenvector pertaining to the highest eigenvalue is worthy of attention in most parameter estimation problems, for this defines the direction of maximum insensitivity, and hence of greatest elongation of the 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 (ie. parameter correlation). The correlated parameters are those whose eigenvector components are significantly non-zero.

The ratio of the highest to lowest eigenvalue constitutes another significant item of information that is forthcoming as a by-product of the parameter estimation process. The square root of this ratio is related to the "condition number" of the matrix that PEST must invert when solving for the parameter upgrade vector - see equation 2.23. If the condition number of a matrix is too high, then inversion of this matrix becomes numerically difficult or even impossible. In the present instance this is an outcome of the fact that solution of the inverse problem approaches nonuniqueness as elongation of the probability ellipsoid increases. In general, if the ratio of the highest to lowest eigenvalue is greater than about 10^8, there is a strong possibility that PEST is having difficulty in calculating the parameter upgrade vector because of parameter insensitivity and/or correlation. Its performance may be seriously degraded as a result.

5.3 Other PEST Output Files

5.3.1 The Parameter Value File

At the end of each optimisation iteration PEST writes the best parameter set achieved so far

(ie. the set for which the objective function is lowest if PEST is running in parameter 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. Example 5.2 illustrates such a file. Note that a PEST parameter value file can be used by program TEMPCHEK in building a model input file based on a template file, by program PESTGEN in assigning initial parameter values to a PEST control file, and by program PARREP in building a new PEST control file from an old PEST control file; see Chapter 10 for further details.

```
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
```

Example 5.2 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 PEST control file; see Section 4.2.2. Then follows a line for each parameter, each line containing a parameter name, its current value and the values of the SCALE and OFFSET variables for that parameter.

5.3.2 The Parameter Sensitivity File

5.3.2.1 The Composite Parameter Sensitivity

Most of the time consumed during each PEST optimisation iteration is devoted to calculation of the Jacobian matrix. During this process the model must be run at least NPAR times, where NPAR is the number of adjustable parameters.

As is explained in Section 2.2.7, based on the contents of the Jacobian matrix, PEST calculates a figure related to the sensitivity of each parameter with respect to all observations (with the latter weighted as per user-assigned weights). The "composite sensitivity" of parameter *i* is defined as:

$$s_i = (\mathbf{J}^t\mathbf{Q}\mathbf{J})_{ii}^{1/2} / m \qquad (5.1)$$

where **J** is the Jacobian matrix and **Q** is the "cofactor matrix"; in most instances the later will be a diagonal matrix whose elements are comprised of the squared observation weights. *m* in equation 5.1 is the number of observations with non-zero weights. Thus 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 pertaining to 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 (ie. the filename base of the current PEST control file). Example 5.3 shows an extract from a parameter sensitivity file.

```
                  PARAMETER SENSITIVITIES: CASE VES4
OPTIMISATION ITERATION NO.  1 ----->
 Parameter_name Group  Current value    Sensitivity    Rel. Sensitivity
   ro1          ro         4.00000         1.25387         0.754905
   ro2          ro         5.00000        0.327518         0.228925
   ro3          ro         6.00000         2.09172          1.62768
   h1           hhh        5.00000        0.176724         0.123525
   h2           hhh        4.00000      4.718210E-02      2.840646E-2


 OPTIMISATION ITERATION NO.  2 ----->
 Parameter_name Group  Current value    Sensitivity    Rel. Sensitivity
   ro1          ro         3.79395         1.30721         0.756995
   ro2          ro         15.0000        0.672146         0.790506
   ro3          ro         4.57028         1.77164          1.16918
   h1           hhh        2.85213        0.661729         0.301198
   h2           hhh        4.00000        0.465682         0.280369
```

Example 5.3 Part of a parameter sensitivity file.

The *relative* composite sensitivity of a parameter is obtained by multiplying its composite sensitivity by the magnitude of the value of the parameter. It is thus a measure of the composite changes in model outputs that are incurred by a *fractional* change in the value of the parameter.

It is important to note that 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. *The relative composite sensitivity of a log-transformed parameter is determined by multiplying the composite sensitivity of that parameter by the absolute log of the value of that parameter.*

As is explained in Section 5.6, composite parameter sensitivities are useful in identifying those parameters which may be degrading the performance of the parameter estimation process through lack of sensitivity to model outcomes. The use of *relative* sensitivities in addition to normal sensitivities assists in comparing the effects that different parameters have on the parameter estimation process when these parameters are of different type, and possibly of very different magnitudes.

Information is appended to the parameter sensitivity file during each optimisation iteration 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 file. In this manner the user is able to track variations in the 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 optimisation iteration to optimisation iteration. Composite parameter sensitivities for any optimisation iteration are calculated using the optimal weight factor (calculated on an iteration-by-iteration basis by PEST) for members of the group "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.

5.3.2.2 Sensitivity Information Recorded on Termination of PEST Execution

At the end of the parameter estimation process (or if PEST is halted prematurely using the "stop with statistics" option), PEST provides a complete listing of composite parameter sensitivities based on the best sensitivity matrix (ie. Jacobian matrix) computed during the optimisation process. "Best" is defined in terms of the aim of the optimisation process; this may be to minimise the objective function (parameter 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).

The point within the parameter estimation process where the "best" Jacobian matrix was computed will vary from run to run. It may have been computed during the last optimisation iteration, or it may have been computed some iterations ago, subsequent attempts to improve the outcome of the optimisation process since that iteration having met with no success. Note also that if there was a marginal improvement in the outcome of the optimisation process during the final optimisation iteration, but not enough to warrant the undertaking of another optimisation iteration, then sensitivities will not correspond exactly to optimised 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 will be a very close approximation to sensitivities calculated for PEST's final parameter estimates. However, if you would like to ensure that sensitivities correspond exactly to optimised parameter values, you can do the following:-

1. Use program PARREP (see Chapter 10) to build a new PEST control file based on optimised parameter values from the present run.

2. Set NOPTMAX in that file to -1, thus requesting that PEST compute sensitivities and then cease execution.

3. Perhaps set the FORCEN variable for each parameter group to "always_3", 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 on the present optimisation run.

Then run PEST.

When writing "completion parameter sensitivities" to the end of the parameter sensitivity file, PEST lists the composite sensitivity and relative 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 5.1, 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 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 working in predictive analysis mode, this weight is ignored by PEST in actual predictive analysis calculations. However it is not ignored in calculating the sensitivity of the sole member of this group to each adjustable parameter for the purpose of recording composite sensitivities pertaining to each observation group at the end of the parameter sensitivity file. The user should consider this when assigning a weight to the sole member of the observation group "predict" when preparing the PEST control file for a predictive analysis run.

When using PEST in regularisation mode, weights assigned to the observation group "regul" are multiplied by the optimal regularisation weight factor determined as part of the parameter estimation process before recording composite sensitivities with respect to the members of this group of each adjustable parameter.

5.3.3 Observation Sensitivity File

The composite observation sensitivity of observation o_j is defined as:

$$s_j = \{\boldsymbol{Q}(\boldsymbol{J}\boldsymbol{J}^T)\}_{j,j}^{1/2} / n \qquad (5.2)$$

That is, the composite sensitivity of observation j is the magnitude of the j'th row of the Jacobian 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. At the end of its run, PEST lists all observation values and corresponding model-calculated values, as well as composite sensitivities for all observations to the "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 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 particularly 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 content. In this case, it may be

possible to omit one of these observations from the parameter estimation 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 parameter estimation process, for this can only be decided in the context of the presence or absence of other observations with similar sensitivities.

Example 5.4 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

Example 5.4 Part of an observation sensitivity file.

5.3.4 The Residuals File

At the end of its execution, PEST writes a "residuals file" listing in tabular form observation names, the groups to which various observations belong, measured and modelled observation values, differences between these two (ie. 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 forms 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. As is explained in Section 2.1.2 of this manual, after the parameter estimation process has been carried out and a value has been obtained for the "reference variance" σ, the standard deviation of each observation can be calculated as the inverse of its weight multiplied by the square root of the reference variance. Care must be taken in interpreting this standard deviation for, being dependent on the fit achieved between model outputs and corresponding field or laboratory measurements, it is a valid measure of observation uncertainty only in so far as the model is a valid simulation of the processes that it is intended to represent.

"Natural weights" as represented on the observation residuals file are the inverse of measurement standard deviations as determined above. If these weights are used in the parameter estimation process, the reference variance will be 1.0.

Where a covariance matrix is supplied for one or more observation groups instead of weights, the residuals file is slightly modified. As well as this, an extra file called a "rotated residuals file" is generated by PEST. See Section 4.3.3 for details.

5.3.5 The Matrix File

During each optimisation iteration, just after it has calculated the Jacobian matrix, if any of the ICOV, ICOR or IEIG variables supplied in the PEST control file are set to 1, PEST calculates the covariance and correlation coefficient matrices, as well as the eigenvalues and normalised eigenvectors of the covariance matrix, for the current set of parameter values. Depending on the settings of the ICOV, ICOR and IEIG variables, these matrices will then be written to a special file named a "matrix file". This file is named *case*.*mtt* where *case* is the current case name (ie. 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 current optimisation iteration only (or, at the end of the parameter estimation process, to the last optimisation iteration).

If any of ICOV, ICOR or IEIG are set to zero, the corresponding matrix is not written to 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 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 to the final matrix file and those recorded to 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 the final optimisation iteration, then he/she may expect that these two sets of matrices will be identical. Nevertheless, there are often differences between these two sets of matrices. These differences result from the fact that the "reference variance" (see equation 2.5) used in the calculation of matrices which are recorded in the matrix file is computed using the objective function calculated at the end of the previous optimisation 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 optimisation iteration, resulting in slight differences between the matrices recorded on the final matrix file and those recorded on the run record file.

5.3.6 Other Files

If requested through the PEST control variable RSTFLE, PEST intermittently stores its data arrays and last two Jacobian matrices in binary files named *case*.*rst*, *case*.*jac* and *case*.*jst*. If PEST execution is re-commenced using the "/r" switch, it reads the first of these binary files in addition to its normal input files; if it is re-started with the "/j" switch it reads all of them.

As is explained in Section 10.6, PEST records the Jacobian matrix corresponding to optimised parameter values to a file named *case*.*jco*. This is accessible by the JACWRIT utility for recording of the Jacobian matrix in text format.

Parallel PEST uses a number of files for communication between PEST and its various

slaves. It also writes a "run management file" documenting the communications history between the various programs taking part in the optimisation process. All of these files are described in detail in Chapter 9.

5.3.7 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 the estimation process at any time.

If you are using the single window version of PEST and the model of which PEST has control writes its own output to the screen, this 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 interrupt PEST execution to inspect the run record file at any time (see the following section). However, if you find it annoying, you may be able to redirect the 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, if program VES cited in the PEST control file of Example 4.2 produces a verbose screen output which is of no real use in the parameter estimation process, the model command line cited in the "model command line" section of the PEST control file could be replaced by

```
ves > temp.dat
```

or

```
ves > nul
```

In the latter case screen output is simply "lost", for there is no *nul* file.

5.3.8 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 will terminate execution with a run-time error message. Unlike PESTCHEK (see Chapter 10), 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 estimation 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. Hence if a run-time error 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 many 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 on the PEST control file.

Another model-related error which can lead to PEST run-time errors of this kind will occur if the subdirectory 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 Example 5.5. If you are using Parallel PEST 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:-

Varying parameter "par1" has no affect on model output -
Try changing initial parameter value, increasing derivative increment,
holding parameter fixed or using it in prior information.
**********************************************************************
```

Example 5.5 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 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 directory 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 single window version of PEST is running, PEST will not respond to the presence of file *pest.stp* until the current model run is complete. Parallel PEST will respond immediately; however the various incidences of the model will continue to run to completion in their own windows after PEST execution has ceased.

While PEST execution is paused, the run record file can be inspected by viewing it using a text editor or viewer from another window.

5.4.2 Restarting PEST with the "/r" Switch

As was discussed in Section 4.2.2, you can instruct PEST to periodically dump the contents of its memory to a number of binary files so that, if its execution is terminated at any stage, it can later be restarted, taking advantage of the work which it has already done. Thus, for example, if you had been using the single window version of PEST and you had previously terminated its execution before the inversion process was complete, you could restart it using the command:-

```
pest case /r
```

where *case* is the filename base of the PEST control file. The restart option is invoked in an identical manner for Parallel PEST; however it may be necessary to restart the slaves first.

When PEST is restarted in this manner, it will not resume execution exactly where it left off; rather it will recommence the parameter estimation process at the beginning of the optimisation iteration in which it was previously interrupted.

In general it is unwise to interfere with any of PEST's input files (ie. the PEST control file as well as its template and instruction files) between interrupting and restarting PEST. While PEST reads all of the data previously contained in its storage arrays from the binary file *case.rst* in the event of a restart, it still needs to obtain the problem dimensions and many of its settings from the PEST control file, the parameter templates from the respective template files and its instructions from the respective instruction files. If any information in any of these files is inconsistent with the information stored in file *case.rst*, PEST's behaviour will be unpredictable.

However, if you are very, very careful, you can alter a number of control variables with impunity. The variables which you may alter are RLAMFAC, PHIRATSUF, PHIREDLAM and NUMLAM which affect the way in which PEST selects Marquardt lambdas, and NOPTMAX, PHIREDSTP, NPHINORED, RELPARSTP and NRELPAR which are termination criteria. You can also alter the derivative variables DERINC, DERINCLB, DERINCMUL and DERMTHD for any parameter group. Thus if, for example, PEST terminates execution with a run-time error message such as shown in Example 5.5, you can edit the PEST control file, altering DERINC, DERINCLB and/or DERINCMUL, and then recommence execution using the restart option.

Program PARREP, one of the PEST utilities described in Chapter 10, provides a much safer means of restarting PEST execution with one or more control variables altered. It allows a new PEST control file to be built from an existing PEST control file and a parameter value file; the latter may contain values optimised by PEST on a previous run. Thus a new PEST

run can be restarted (with or without altered control settings) where an old one left off.

5.4.3 Restarting PEST with the "/j" Switch

PEST can also be restarted with the "/j" switch; this is an integral part of the user-interaction functionality provided by PEST. It is discussed in Section 5.6.

5.5 If PEST Won't Optimise

5.5.1 General

PEST allows the user to follow closely the progress of an optimisation run both through its screen output and through the user's ability to inspect the run record file. Through watching the value of the objective function (referred to as "phi" on the PEST run record) from optimisation iteration to optimisation iteration, you can monitor PEST's ability and efficiency in lowering the objective function to the minimum which can be achieved within the user-provided parameter domain.

There can be many reasons for a failure on the part of PEST to lower the objective function; in most cases the problem can be easily overcome by adjusting one or a number of PEST input variables. The fact that PEST provides so many control variables by which it can be "tuned" to a particular model is one of the cornerstones of its model-independence. In other cases, PEST's progress can be assisted by selectively holding either one or a few parameters at their current values; the user may then re-commence PEST execution at that spot at which the Jacobian matrix was last calculated in order to re-compute the last parameter upgrade vector, or simply continue execution with the selected parameters held fixed for a while. See the next section for details.

If you are using a particular model for the first time with PEST, you may wish to run a theoretical case first. You should use the model to fabricate a sequence of observation values of the same type as that for which you have laboratory or field measurements, and then use these fabricated observations as your field or laboratory data. Then you should run PEST, using as your initial parameter estimates the parameters that gave rise to the fabricated observation set. PEST should terminate execution after the first model run with an objective function value of zero. (In some cases it will not be exactly zero because of roundoff errors; nevertheless it should be extremely small.) In this way you can check that PEST is supplying correct parameter values to the model, running the model correctly, and reading observation values correctly.

Next you should vary the parameter initial values and run PEST again. It is at this stage, while working with a theoretical dataset for which you know PEST should achieve a low objective function value, that you can adjust PEST control variables in order to tune PEST to the model. Note that it is unlikely that you will achieve an objective function value of zero again; though, depending on the number of observations and their magnitudes and weights, the objective function should nevertheless be reduced to as close to zero as roundoff errors permit (provided the model is not beset by severe nonlinearities and/or the presence of local objective function minima). In most cases PEST is able to solve a parameter estimation problem using substantially less than 20 optimisation iterations.

If PEST does not lower the objective function, or lowers it slowly, you should run through the following checklist of reasons for PEST's poor performance. In most instances the problem can be rectified.

5.5.2 Derivatives are not Calculated with Sufficient Precision

Precise calculation of derivatives is critical to PEST's performance. Improper derivatives calculation will normally be reflected in an inability on the part of PEST to achieve full convergence to the optimal parameter set. Sometimes, in such circumstances, PEST will commence an optimisation run in spectacular fashion, lowering the objective function dramatically in the first optimisation iteration or two. But then it "runs out of steam", failing to lower it much further.

Make sure that model outcomes are being written to the model output file with the maximum precision which the model allows. If the model places an upper limit on output precision, ensure that the parameter increments used for derivatives calculation are large enough to cause a useable change in all model-calculated observations, given the number of significant digits in which they are expressed. Try not to make parameter increments too large though, or finite-difference-generated derivatives will be a poor approximation to the real thing. However if they must be large, use one of the three-point methods of derivatives calculation. Try the "parabolic" method first; if that doesn't work, use the "best-fit" method.

Check that parameter values are written to model input files with enough significant digits to reflect parameter increments. If some parameters become very low as the optimisation process progresses, you may need to provide a suitable lower bound on derivative increments through the parameter group variable DERINCLB, or calculate the increment using the largest member of a parameter group by denoting the group variable INCTYP as "rel_to_max".

For a model which solves large matrix equations using an iterative method, you should ensure that the model's solution convergence criterion is set very low so that model-generated observations are calculated with a high degree of precision. However if you set it too low the model solution procedure may not converge; worse still, it may converge for some parameter sets and not for others. To overcome this you may need to make a small change to the model such that it prints out its solution vector even if it has not converged according to your very stringent convergence setting; alternatively you could employ a batch process such as was demonstrated in Example 4.3.

5.5.3 High Parameter Correlation

There is often a temptation in fitting models to data, to improve the fit between modelled and measured observations by increasing the number of adjustable parameters. While it is true that this can result in a lowered objective function, it is not always true that such an improvement increases a model's ability to make reliable predictions, or that a high number of parameters represents a valid interpretation of the dataset to which the model's outcomes are matched. Furthermore, as the number of parameters requiring estimation is increased, there will come a stage where PEST's ability to lower the objective function by adjusting the values of these parameters is diminished due to the effects of roundoff error (particularly for highly nonlinear models); this applies not just to PEST but to any parameter estimation

package.

The trouble with increasing the number of parameters without limit is that, sooner or later, some parameters become highly correlated. This results from the fact that the measurement set upon which the parameter estimation process is based may not have the ability to discriminate between different combinations of parameter values, each combination giving rise to an equally low objective function. As has already been discussed, the extent to which parameter pairs and/or groups are correlated can be gleaned from an inspection of the correlation coefficient and/or eigenvector matrices.

If parameters are too highly correlated the matrix $\mathbf{J}^t\mathbf{QJ}$ of equation 2.18 becomes singular. However because PEST adds the Marquardt parameter to the diagonal elements of this matrix before solving for the parameter upgrade vector (see equation 2.20), rendering it singular no longer, an upgrade vector can nevertheless be obtained. Eventually, unless circumvented by roundoff errors, an objective function minimum will be obtained through the normal iterative optimisation process. However the parameter set determined on this basis may not be unique. Hence, if you are running a theoretical case, PEST may determine a parameter set which is entirely different from the one which you used to generate the artificial measurement set. In spite of this, the objective function may be very small.

In addition to the nonuniqueness problem, the optimisation process may become very slow if there are many parameters in need of estimation. There are two reasons for this. The first is that PEST requires at least as many model runs as there are adjustable parameters in order to fill the Jacobian matrix during each optimisation iteration. The second reason is based on the possible near-singular condition of the normal matrix and the way in which PEST adjusts the Marquardt lambda upwards in response to this. In general, while high lambda values can lead to a rapid lowering of the objective function at the early stages of the parameter estimation process when parameter values are far from optimal, it is normally far better to decrease lambda as the objective function minimum is approached. As discussed in Section 2.1.5, use of a high Marquardt lambda is equivalent to the use of the "steepest descent" optimisation method; however this method is notoriously slow when parameters are highly correlated, due to the phenomenon of "hemstitching" as the parameter upgrade vector oscillates across narrow objective function valleys in parameter space. But if lambda cannot be lowered because the normal matrix would then become singular, or at best ill-conditioned, due to the excessive number of parameters requiring estimation, there will be no way to prevent this.

The incorporation of prior information into the estimation process can often add stability to an over-parameterised system. Likewise, removing a number of parameters from the process by holding them fixed at strategic values may yield dramatic improvements in PEST's performance. In many environmental modelling contexts a spectacular increase in PEST's ability to estimate large numbers of parameters can be achieved by running PEST in regularisation mode; see Chapter 7 for full details.

5.5.4 Inappropriate Parameter Transformation

PEST allows adjustable parameters to be either log-transformed or untransformed. A suitable choice for or against log transformation for each parameter can make the difference between a successful PEST run and an unsuccessful one.

Trial and error is often the only means by which to judge whether certain parameters should be log-transformed or not. There is no general rule governing which parameters are best log-transformed; however experience has shown that parameters whose values can vary over one or a number of orders of magnitude often benefit from log transformation. Log transformation of these parameters will often linearise the relationship between them and the observations, making the optimisation process more amenable to the linearity assumption upon which the equations of Chapter 2 are based.

Use of suitable SCALE and OFFSET variables can be used to change the domain of a parameter such that logarithmic transformation, and with it the possible benefits of increased linearity, becomes a possibility. The use of parameter scaling and offsetting is discussed in Sections 2.2.4 and 4.2.4.

More complex parameter transformations than logarithmic which may, in some circumstances, decrease the nonlinearity of a particular parameter estimation problems can be undertaken using the parameter preprocessor PAR2PAR; see Chapter 10 for details.

5.5.5 Highly Nonlinear Problems

If the relationship between parameters and observations is highly nonlinear, the optimisation process will proceed only with difficulty. As discussed above, such nonlinearity may sometimes be circumvented through appropriate transformation of some parameters. However, in other cases this will make little difference. In such cases the Gauss-Marquardt-Levenberg method of parameter estimation on which PEST is based may not be the most appropriate method to use.

Sometimes the use of a high initial Marquardt lambda is helpful in cases of this type. Also, the relative and absolute parameter change limits (RELPARMAX and FACPARMAX on the PEST control file) may need to be set lower than normal; a careful inspection of the PEST run record file may suggest suitable values for these variables and, indeed, which parameters should be relative-limited and which should be factor-limited. Parameter increments for derivatives calculation should be set as low as possible without incurring roundoff errors. The three-point "parabolic" method may be the most appropriate method for calculating derivatives because of its quadratic approximation to the relationship between observations and parameters. The incorporation of prior information into the parameter estimation process (with a suitably high weight assigned to each prior information equation) may also yield beneficial results.

5.5.6 Discontinuous Problems

The equations derived in Chapter 2, upon which the Gauss-Marquardt-Levenberg algorithm is based, are predicated on the assumption that observations are continuously differentiable functions of parameters. If this assumption is violated for a particular model, PEST will have extreme difficulty in estimating parameters for that model. (However, it may have some success if the dependence is continuous, if not continuously differentiable.)

5.5.7 Parameter Change Limits Set Too Large or Too Small

As discussed above with respect to highly nonlinear problems, a suitable choice for relative

and factor parameter change limits (ie. RELPARMAX and FACPARMAX) may allow optimisation to be carried out under hostile circumstances. However if these change limits are set too low, minimisation of the objective function may be hampered as the upgrade vector is continually shortened in order to conform to the demands of these limits. An inspection of the run record file should reveal immediately whether parameter upgrades are being limited by these variables. If the maximum relative and/or factor parameter changes per optimisation iteration are consistently equal to the respective user-supplied limits, then it is possible that these limits could be increased; however, if your model is highly nonlinear or "messy", it may be better to keep RELPARMAX and FACPARMAX low as they may prevent parameter adjustment "overshoot".

You should exercise caution in choosing which parameters are relative-limited and which are factor-limited. Remember that if a parameter is factor-limited, or if it is relative-limited with a limit of less than 1, the parameter can never change sign. Conversely, if a parameter is relative-limited with a limit of 1 or greater, it can be reduced right down to zero in a single step without transgressing the limit; this may cause parameter "overshoot" problems for some nonlinear models and a factor limit may need to be considered. However the latter cannot be used if the parameter can change sign. Faced with quandaries of this type, the parameter OFFSET variable may be useful in shifting the parameter domain such that it does not include zero.

As described in Section 5.6, RELPARMAX and FACPARMAX can be altered midway through a PEST optimisation run. Furthermore, if the parameter adjustment vector is dominated by a particular insensitive parameter such that the change to that parameter is at its RELPARMAX or FACPARMAX limit and the changes to other parameters are minimal, then the offending parameter can be held at its current value through the user-intervention process described in Section 5.6.

5.5.8 Poor Choice of Initial Parameter Values

In general, the closer are the initial parameter values to optimal (ie. the values for which the objective function is at its global minimum), the faster will PEST converge to that global minimum. Furthermore not only does a wise choice of initial parameter values reduce the PEST run time, it may also make optimisation possible, especially for highly nonlinear models or models for which there are local objective function minima at places removed in parameter space from the location of the global objective function minimum.

5.5.9 Poor Choice of Initial Marquardt Lambda

The PEST algorithm is such that PEST should find its way to a close-to-optimal Marquardt lambda at each stage of the parameter estimation process. However if you supply an initial Marquardt lambda which is far from optimal, the adjustment to optimal lambda may not occur. After attempting a parameter upgrade with the initial lambda, PEST searches for alternative lambdas, using the input variable RLAMFAC to calculate them. If the initial lambda was poor, these alternative lambdas may be little better in terms of lowering the objective function than the first one. Soon, in accordance with the settings provided by PHIREDLAM and NUMLAM on the PEST control file, PEST may move on to the next optimisation iteration, having achieved little in lowering the objective function. The story

may then be repeated at the next optimisation iteration, and perhaps the next as well. Soon, because the objective function has not been lowered (or has been lowered very little) over a number of iterations, PEST will terminate execution in accordance with one of its termination criteria.

In most cases, the choice of an initial Marquardt lambda of between 1.0 and 10.0 works well. Nevertheless, if PEST spends the first few optimisation iterations adjusting this to a vastly higher value (or a vastly lower value - but remember that lambda is reduced in the normal course of the optimisation process anyway) before making great gains in objective function reduction, then you will probably need to reconsider your choice of initial lambda in subsequent uses of PEST in conjunction with the same model. However, if the parameter estimation process simply does not "get off the ground", you should start again with an entirely different lambda; try a much greater one first, especially if PEST has displayed messages to the effect that the normal matrix is not positive definite.

To help PEST search farther afield for a suitable Marquardt lambda, perhaps you should set the input variable RLAMFAC high for a while. However it is bad practice to keep it high through the entirety of an optimisation run; hence if PEST finds a lambda which seems to work you should terminate PEST execution, supply that lambda as the initial lambda, reset RLAMFAC to a reasonable value (eg. 2.0) and start the optimisation process again.

Experience has shown that if the initial parameter set is poor, PEST may need a high Marquardt lambda to get the parameter estimation process started. Also the Marquardt lambda may need to be greater for highly nonlinear problems than for well-behaved problems.

Note that the Marquardt lambda is one of the input variables that can be adjusted in mid-run through user-intervention; see Section 5.6.

5.5.10 Observations are Insensitive to Initial Parameter Values

For some types of models, observations can be insensitive to initial parameter values if the latter are not chosen wisely. For example, if you wish to optimise the resistivities and thicknesses of a three-layered half-space on the basis of electric current and voltage measurements made on the surface of that half-space, it would be a mistake to provide all the layer resistivities with the same initial value. If you did, the model would be insensitive to the thicknesses of either of the upper layers (the lowest layer extends to infinity) because the half-space is uniform no matter what these thicknesses are. Hence PEST will set about calculating derivatives of observations with respect to these thicknesses and discover that the derivatives of all observations with respect to all thicknesses are zero. It will then issue a message such as "parameter "h1" has no effect on model output". This problem can be easily circumvented by choosing initial layer resistivities which are different from each other.

In other cases the solution may not be as obvious. For some models, a certain parameter may have very little effect on model outcomes over part of its domain, yet it may have a much greater effect on these outcomes over other parts of its domain. If the optimised value lies within the insensitive area, a large degree of uncertainty will surround its estimate. However if the optimal value lies in the sensitive part of the parameter's domain it is likely that the parameter will be well-determined (unless, of course, it is highly correlated with some other

parameter). In either case you should take care to ensure that the number which you supply as the parameter's initial value is within the sensitive part of its domain.

5.5.11 Parameter Cannot be Written with Sufficient Precision

In certain unusual cases an optimal parameter value may not be capable of representation in a field of limited width on the model input file. The obvious solution to this problem is to increase the width of the parameter field in the corresponding template file. However this may not be possible if model input format requirements are too rigid.. The only other remedial action that can be taken is to set the DPOINT variable to "nopoint", thus allowing a gain of one extra significant figure in some circumstances. Before you do this, however, make sure that the model can still read the parameter value correctly with its decimal point omitted. (Note that PEST omits the decimal point only if it is redundant. However because some models may use format specifiers which make certain assumptions about where an absent decimal point should be, it is possible that a number lacking a decimal point may be misinterpreted; see Section 3.2.6 .)

5.5.12 Incorrect Instructions

If the objective function cannot be lowered it is possible that PEST is reading the model output file incorrectly. If a non-numeric string is present on the model output file at a place where PEST has been lead to expect (through its instruction set) a number, PEST will terminate execution with an appropriate error message. However if a number is present where PEST expects one, but this number is not the one that you intended PEST to read when you built the instruction set, then it is unlikely that PEST's performance in lowering the objective function will be good.

You can check that PEST is reading the correct numbers by terminating PEST execution using the "Stop with statistics" option. PEST will then list on its run record file the model-generated observations corresponding to its best parameter set achieved so far. As PEST also lists the residual corresponding to each model-generated observation, an incorrectly read model outcome may be apparent as that for which there is an unusually high residual.

Note that program INSCHEK can also be used to check that an instruction file does, indeed, read the correct numbers from a model output file. See Chapter 10.

5.5.13 Upgrade Vector Dominated by Insensitive Parameters

This is a common cause of poor PEST performance. It is discussed in greater detail in the following section.

5.6 User Intervention

5.6.1 An Often-Encountered Cause of Aberrant PEST Behaviour

Where many parameters are being estimated and some are far more insensitive than others, it is not uncommon to encounter problems in the parameter estimation process. 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 with other, more sensitive, parameters; this large adjustment of insensitive parameters may be necessary if alterations to their values are to have any effect on the objective function. However the magnitude of the change that can be incurred by any parameter during any particular optimisation iteration is limited by the values assigned to the PEST control variables RELPARMAX and FACPARMAX. PEST reduces the magnitude (but not the direction) 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, restricting the magnitude of the upgrade vector such that the change to the value of the insensitive parameter is limited to RELPARMAX or FACPARMAX (depending on its PARCHGLIM setting) will result 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 and FACPARMAX is not necessarily the solution to the problem, for parameter change limits are necessary in order 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 either the maximum relative parameter change or the maximum factor parameter change for a particular optimisation iteration (as printed to the screen and to the run record file) is equal to RELPARMAX or FACPARMAX respectively, and that the objective function is reduced very little. PEST records the names of parameters that have undergone the largest factor and relative changes at the end of each optimisation iteration. More often than not, an inspection of the parameter sensitivity file (see Section 5.3.2) will reveal that these same parameters possess a low sensitivity.

5.6.2 Fixing the Problem

The solution to the above 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.

5.6.3 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 optimisation iteration.

A parameter hold file is shown in Example 5.6.

```
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
```

Example 5.6 Part of 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 three PEST control variables using the parameter hold file. These are RELPARMAX, FACPARMAX and LAMBDA. The syntax is shown in Example 5.6, ie. the name of the variable must be followed by its new value. *It is important to note that if a parameter hold file is left "lying around", any lines altering the value of lambda should be removed or "commented out" or PEST will be prevented from making its normal adjustment to lambda from iteration to iteration. This may severely hamper the optimisation 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 Example 5.6, ie. 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 optimisation iterations.

The format for the sixth line in Example 5.6 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 hold any parameter in the named parameter group temporarily fixed if the sensitivity of that parameter is less than the supplied number (ie. *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 7th line of Example 5.6, 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 8th line of Example 5.6 is:-

```
hold eigenvector n highest m
```

This option has been included in PEST's parameter holding functionality to accommodate 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. As is discussed in Chapter 2, 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 holding command, eigenvectors 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 (ie. 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 5.5, eigenvalues and eigenvectors are available at all stages of the parameter estimation process through the matrix file recorded by PEST during every optimisation iteration.

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 causing problems in the inversion 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 were 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 minimised.

5.6.4 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 optimisation iteration has elapsed. Even if it has met with little success in lowering the objective function, PEST moves on to the next optimisation iteration commencing, once again, its time-consuming calculation of the Jacobian matrix (possibly having switched to the use of three-point derivatives).

However the functionality exists within PEST to halt its execution at any time and restart it at that place at which it last commenced calculation of the parameter upgrade vector, ie. 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 Jacobian matrix in a binary file each time it is calculated; the Jacobian matrix is easily retrieved if PEST is asked to re-calculate the parameter upgrade vector.

Re-commencement of PEST execution for upgrade vector re-calculation is effected by running PEST using the command

```
pest case /j
```

or, if using Parallel PEST,

```
ppest case /j
```

where *case* is the current PEST case name, ie. the filename base of the PEST control file; "j" stands for "Jacobian". 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 either within the same optimisation 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 is 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, in some over-parameterised cases, a user can progressively hold more and more parameters fixed until a significant improvement in the objective function is realised. Then he/she can let PEST move on to the next optimisation iteration.

5.6.5 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. It also records the names of the parameters undergoing these maximum changes. This, in combination with the contents of the parameter sensitivity file, may assist the user in deciding which (if any) parameters to temporarily hold at their present values using directives supplied in the parameter hold file.

5.7 PEST Postprocessing

5.7.1 General

After the parameter estimation process is complete, a thorough examination must be undertaken of the results of this process before the estimated parameters can be used in a calibrated model which is run for predictive purposes. This section presents a guide to some of this analysis procedure. The discussion necessarily pertains to general parameter estimation post-processing; extra, case-specific analysis will be required for each particular instance of PEST deployment.

The information upon which to base post-processing analyses such as those outlined in this section is contained in the PEST run record file and in a number of other files that are written by PEST, often in a format that facilitates plotting using commercial graphing and spreadsheet packages. See Sections 5.2 and 5.3 for details.

5.7.2 Parameter Values

Parameter values should be inspected for reasonableness. It is often a good idea to undertake initial PEST runs with parameter bounds set very wide. In this way PEST is free to assign unrealistic values to certain parameters if this is required in order to achieve goodness of fit between model outcomes and corresponding field data. When reasonable parameter bounds are then imposed it will often be possible to obtain just as good a fit, this being an outcome of the parameter correlation that is a feature of most real-world model calibration. However if it is not possible to achieve just as good a fit with parameters constrained to reasonable values, then PEST is providing valuable information on model adequacy, for this state of affairs indicates that the model may not be simulating all aspects of the system that it is intended to portray. Whether this is a worrying matter or not depends on the particular modelling application. It is sometimes valid modelling practice to allow unsimulated subprocesses to be represented by out-of-range surrogate parameter values. In other applications this is completely unacceptable. The making of the appropriate decision in each case is part of the art of modelling.

5.7.3 Parameter Statistics

This manual discusses at length the role of the various statistics that are produced as an outcome of the PEST parameter estimation process. These include the parameter covariance matrix, correlation coefficient matrix, covariance matrix eigenvalues and eigenvectors and parameter confidence intervals. Though all of these quantities are only approximate because

of the fact that their calculation is based on a linearity assumption that is often violated, there is nevertheless much to be learned about the model, its parameterisation, and its appropriateness for a specific application from an inspection of these quantities.

High levels of parameter uncertainty can result from a poor fit between model outcomes and field observations, from a high level of parameter correlation, from insensitivity on the part of certain parameters, or from all of these. Provided PEST has not faltered in the nonlinear parameter estimation process (see Section 5.5), the first condition indicates that either field data is poor or that the model is inappropriate. (Or it may indicate that certain model parameters that were held fixed during the optimisation process may have been held at inappropriate values.) Fortunately, the reason for the poorness of fit is often easy to identify, though not necessarily easy to rectify (especially if it requires alterations to model design).

Parameter uncertainty resulting from high levels of parameter correlation may or may not be a defect, depending on the problem to which the model will be applied. Highly correlated parameters are recognised through an inspection of the correlation coefficient matrix, and through an inspection of the eigenvalues and eigenvectors of the parameter covariance matrix. Whether indeterminacy of these parameters affects a particular model prediction depends on the prediction. The ultimate test of this is to use PEST's predictive analyser (see Chapter 6) in order to test the range of predictive variability that occurs as an outcome of parameter uncertainty. As a general rule, if predictions that are to be made by the model are of the same type as the measurements used in the calibration process, and if the "stress regime" imposed on the system represented by the model under predictive conditions is not too different from that prevailing under calibration conditions, then the effect of correlation-induced parameter uncertainty on predictive uncertainty may not be too large. However if any of these conditions are violated, predictive uncertainty may be very large indeed.

Parameter uncertainty caused by parameter insensitivity is often difficult to rectify. Recall that composite parameter sensitivities are listed in the parameter sensitivity file produced by PEST. Parameter insensitivity results from the fact that the dataset used in the optimisation process simply does not possess the information content that is required to resolve the values of offending parameters. Thus these parameters can assume a range of values with minimal effect on model outcomes.

Once again, parameter uncertainty arising out of parameter insensitivity may or may not result in high levels of predictive uncertainty. As always, the extent of predictive uncertainty can be estimated using PEST's predictive analyser.

Parameter uncertainty resulting from either excessive correlation or from insensitivity of parameters can often be reduced by including more measurement data in the inversion process, especially if these measurements are "targeted" at the offending parameters. Extra information in the form of prior information can also be very effective in increasing parameter sensitivity and in reducing parameter correlation. Use of PEST in regularisation mode can also be very effective in reducing the deleterious effects of parameter insensitivity.

5.7.4 Residuals

An analysis of the differences between model outcomes and corresponding field or laboratory data is an extremely important part of any parameter estimation application.

The mathematical basis of the parameter estimation algorithm used by PEST relies on the assumption that measurement uncertainties are uncorrelated, ie. that the uncertainty associated with any one measurement is unrelated to that associated with any other measurement. If measurements do, in fact, exhibit correlation then an observation covariance matrix should be used in place of observation weights so that the "rotated residuals" are uncorrelated. See Section 4.3 for details.

One of the first tasks that should be undertaken after the parameter estimation process is complete is to determine whether residuals (or rotated residuals) are, in fact, randomly distributed about a mean of zero with minimal correlation between them. PEST provides some degree of assistance in making this determination. As described in Section 5.2.8 and in Section 4.3, on its run record file PEST lists a number of pieces of information pertaining to the residuals taken as a whole, and to the residuals pertaining to each observation group.

Much can be learned about residuals (and about the efficacy of the parameter estimation process that gave rise to these residuals), by different types of graphical inspection. Graphs can be readily obtained using commercial plotting software based on the information contained in the residuals file, and perhaps the rotated residuals file, written by PEST at the end of its run. These files can also be used as a basis for more sophisticated residuals analysis using commercial statistical software.

Ideally, a plot of weighted (rotated) residuals against weighted or unweighted (rotated) observation values (or weighted or unweighted simulated values) should reveal no dependence of one upon the other (unless weights were purposefully chosen to accentuate certain observation types). Furthermore, a plot of weighted (rotated) residuals against the normal variate should (ideally) reveal that residuals are normally distributed.

Perhaps the most important types of residuals analyses are those that can only be undertaken in a case-specific manner. For example, if the model being calibrated is a steady-state ground water model and the measurements are of water levels in various bores spread throughout the model domain, then it is important that residuals be plotted at the locations of their respective boreholes and superimposed on a map of the area; "proportional posting" and/or contouring of these residuals may also be useful. Such a two-dimensional graphical portrayal of residuals will immediately indicate any spatial correlation between them. If such spatial correlation exists, this is an indication that the manner in which the model domain has been subdivided for parameterisation purposes could do with some improvement.

If measurements used in the calibration process represent the variation of some quantity over time at one or a number of measurement sites, then superimposed plots of model-generated and measured quantities against time, or of the residuals themselves against time, at all measurement sites should be inspected. Any tendency of the model to overpredict or underpredict over extended periods of time, or over certain segments of the graphs, should be noted, for this may indicate an inadequacy in the model's ability to represent facets of real-world behaviour; such an inadequacy may have unwanted repercussions when the model is used for predictive purposes. If time-varying measurements are made at different geographical locations, plots of the spatial distribution of residuals at different times may also reveal worrying departures from independent behaviour, spatial correlation possibly indicating, once again, certain inadequacies on the part of the model.

5.7.5 Over-Parameterisation

In many cases of model deployment the fit between model outcomes and field measurements can be improved with relative ease by declaring more parameters as adjustable, or by simply adding more parameters to the model. This is particularly the case for distributed parameter models where it is an easy matter to undertake a finer subdivision of the model domain for parameterisation purposes, thus endowing the model with a greater number of parameters that require estimation. Ultimately, through adding more and more parameters, it may be possible to reduce the objective function to almost zero as every nuance of system behaviour is replicated by the model outputs.

It is very important to be aware of the fact that a good fit under calibration conditions does not guarantee an accurate model prediction. In general, the more parameters that are estimated, the more highly are they correlated, and the more likely it is that some of them are insensitive. Both of these will contribute to a high degree of parameter uncertainty which may result in a high degree of uncertainty for at least some types of model predictions.

If the system under study is such that a high level of parameterisation is nevertheless required because it is necessary that the model be capable of reproducing the "fine detail" of system response, then the user should be very aware of the uncertainty surrounding estimated parameters and of the uncertainty that is likely to accompany many of the predictions made by the model. In this case predictive analysis is a necessity. Alternatively, some kind of problem "regularisation" is required, this being a mechanism by which parameter uncertainty can be reduced through the enforcement of known or suspected relationships between parameters (such as a "smoothing condition" in the case of distributed parameter models). Such techniques can be very effective in combating the deleterious effects of over-parameterisation; if properly designed they can be such as to ensure that departures from simplicity in parameterisation are limited to those that are just sufficient to reproduce the required level of detail in system behaviour. See Chapter 7 for a description of PEST's uses in "regularisation mode" for more details.

5.7.6 Covariance Matrix for Best-Fit Parameters

The user is reminded that once PEST has calculated an optimal set of parameter values and, in accordance with one of its termination criteria, finishes execution, it calculates the covariance matrix and the statistics derived therefrom on the basis of the Jacobian matrix giving rise to the best set of parameter values, unless these were achieved on the very last optimisation iteration. If this is the case, then PEST's termination criteria are such as to ensure that the statistics calculated on the basis of the "not-quite-optimal" parameter set will depart only minimally from those calculated on the basis of the optimal parameter set. Use of the "not-quite-optimal" parameter set saves PEST from having to carry out another set of model runs at the end of the optimisation process in order to re-calculate the Jacobian matrix on the basis of best-fit parameters. Normally this is quite acceptable. However if you want to be absolutely sure that the covariance matrix and its derived statistics are as right as they can be (notwithstanding the fact that they are based on an often poorly-met linearity assumption), then you way wish to undertake a special PEST run simply to calculate these statistics on the basis of optimal parameter values. You can do this by following these steps:-

1. Use program PARREP (see Section 10.5) to build a new PEST control file in which

initial parameter values are actually optimised parameter values determined on a previous PEST run.

2. Set the control variable NOPTMAX (see Section 4.2.2) to -1 in the new PEST control file.

3. Run PEST.

Greater accuracy in derivatives calculation (and hence greater accuracy in calculation of the parameter covariance matrix) can be achieved with the FORCEN variable (see Section 4.2.3) for all parameter groups set to "always_3" in the new PEST control file.

5.7.7 Model Outputs based on Optimal Parameter Values

Whether or not you have undertaken the steps outlined in the previous section for obtaining statistics based on optimal parameters, the last model run undertaken by PEST prior to termination of execution will not normally have been undertaken on the basis of optimal parameters. (If the instructions outlined in the previous section are followed, the last model run will, in fact, have been undertaken with one parameter value slightly incremented or decremented from optimality for the purpose of derivatives calculation.) Hence model output files available at the termination of PEST execution will not contain "best-fit model outputs". These can only be obtained if a model run is explicitly undertaken on the basis of optimised parameter values. This can be achieved in either of two ways. The first is to use TEMPCHEK to generate a set of model input files based on template files for the current case together with the parameter value file produced by PEST in the course of its previous parameter estimation run; see Section 10.1 for details. The second option is to use PARREP in the manner described in the previous section, but with NOPTMAX set to 0 in the new PEST control file so that PEST undertakes only one model run, this being for the purpose of objective function calculation.

6. Predictive Analysis

6.1 The Concept

6.1.1 What Predictive Analysis Means

In the discussion that follows, reference will be made to PEST's usage in the role of model calibration. However PEST is often used in the role of data interpretation as well, particularly geophysical data interpretation in which earth properties are inferred from a number of discrete surficial or downhole measurements. Though not referenced specifically, the following discussion is just as applicable to PEST's usage in that role as it is to PEST's usage in model calibration.

PEST "calibrates" a model by reducing the discrepancies between model outputs and field observations to a minimum in the weighted least squares sense. The differences between field measurements and model outputs are encapsulated in an "objective function" defined as the weighted sum of squared deviations between field observations and corresponding model outputs. As PEST executes, it progressively reduces this objective function until it can reduce it no more.

In many cases the "landscape" of the objective function in parameter space is not comprised of a discrete bowl-shaped depression with the objective function minimum lying neatly at the bottom of that depression. Rather (especially if parameters number more than just a few), the objective function minimum often lies at the bottom of a long, narrow valley of almost equal depth along its length. Any parameters for which the objective function lies within the valley can be considered to calibrate (or almost calibrate) the model. Figure 6.1 illustrates this situation for a simple, linear, two-parameter model; for a linear model, contours of the objective function in parameter space are always elliptical. The situation in a more complex nonlinear case is schematised in Figure 6.2. In both of these figures, p_1 and p_2 are the values of the two parameters.

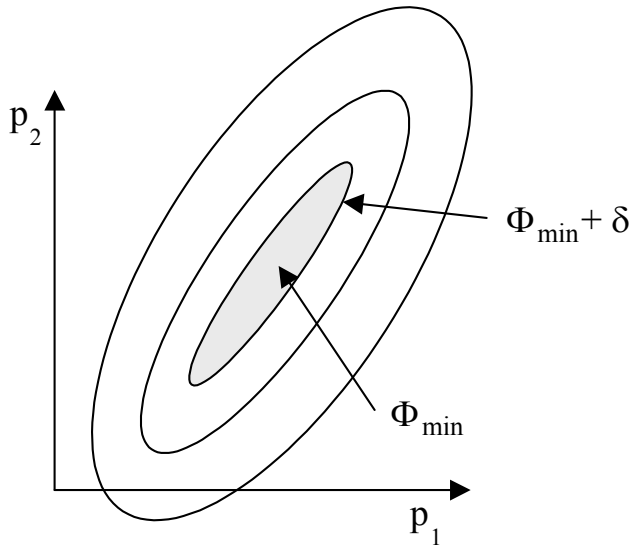


Figure 6.1. Objective function contours in parameter space; linear model.

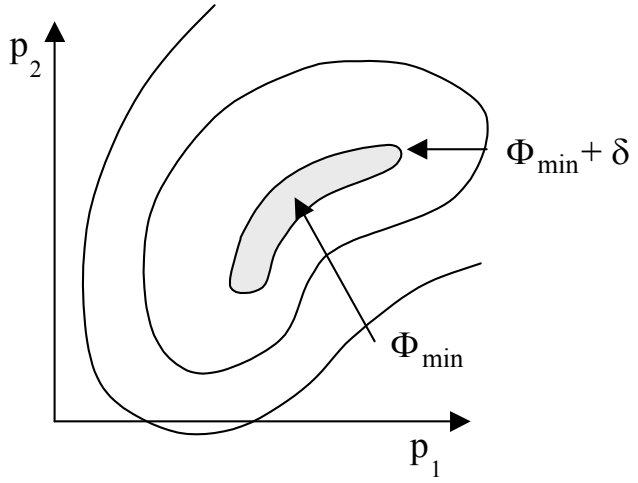


Figure 6.2. Objective function contours in parameter space; nonlinear model.

Both of Figures 6.1 and 6.2 depict situations where there is a high degree of parameter correlation – that is, one parameter can be varied in harmony with another with virtually no effect on the objective function. Thus the solution to the inverse problem (ie. the model calibration problem) is nonunique. If the minimum of the objective function is denoted as Φ_{min}, and if all parameters for which the objective function is less than $\Phi_{min} + \delta$ (where δ is relatively small) can also be considered to calibrate the model, then the range of parameter values which can be considered to calibrate the model can be quite large indeed. In Figures 6.1 and 6.2 the set of "allowable parameter values" (from the model calibration point of view) occupies the area that lies within the $\Phi_{min} + \delta$ contour in parameter space.

In most instances of model calibration, only a single set of parameters lying within the $\Phi_{min} + \delta$ contour of Figures 6.1 and 6.2 is calculated. Model predictions are then made with this single parameter set. An obvious question is this: what would have been the model's

predictions if another set of parameters lying within the $\Phi_{min} + \delta$ contour were used for predictive purposes instead, particularly if this alternative set is relatively distant from the original parameter set as measured in parameter space?

6.1.2 Some Solutions

A number of different methods are available for answering this question. However traditional "sensitivity analysis" cannot be used. The term "sensitivity analysis" is often used to describe the technique whereby parameter values are individually varied from calibration values in order to determine the effects of these changes on model predictive outcomes. This is an unacceptable method of predictive analysis in most instances because unless parameters are varied in certain discrete ratios (that vary with parameter value for nonlinear models) the model is immediately uncalibrated as soon as any parameter is varied from its calibrated value. To fully explore the repercussions of parameter nonuniqueness on predictive nonuniqueness, parameters must be varied in such a way that the objective function hardly changes. As is apparent from Figures 6.1 and 6.2, parameter values can often vary enormously while the objective function changes very little. If parameters are simply incremented and/or decremented one by one, and the effect of this variation tested on both model predictions and on the objective function calculated under calibration conditions, it is likely that the latter will rise very quickly, giving the modeller the false impression that parameter values are estimated with a high degree of precision by the calibration process, and hence that predictive nonuniqueness resulting from parameter nonuniqueness will be slight because of the absence of the latter.

Monte-Carlo analysis is often used to examine uncertainty in model predictions. Parameter sets can be generated at random; for each such parameter set the model is run under calibration conditions. If the resulting objective function is above $\Phi_{min} + \delta$ the parameter set is rejected. If it is below $\Phi_{min} + \delta$ the model is then run under predictive conditions. After many thousands of model runs have been undertaken a suite of predictions will have been built up, all generated by parameter sets which satisfy calibration constraints. In many cases some kind of probability distribution can then be attached to these predictions, based on where corresponding calibration objective functions lie with respect to Φ_{min} and the $\Phi_{min} + \delta$ contour.

This method of predictive analysis has many attractions; however its main disadvantage is in the number of model runs required. Where there are any more than a handful of adjustable parameters, the dimensionality of the problem requires that millions of model runs be undertaken, rendering the method intractable in many practical settings.

6.1.3 The "Critical Point"

The effect of parameter nonuniqueness on predictive nonuniqueness depends on the prediction being considered. In many instances a model is calibrated under a very different stress regime from that under which it will operate to make predictions; in other cases the stress regimes will be similar. In some cases model predictions will be of the same type as those that were used for calibration; in other cases a model will be required to generate predictions of a very different type from those used in the calibration process. In some cases predictions will be required at the same locations as those for which data was available to

assist in the calibration process; in other cases predictions will be required at very different places.

In general, the more similarity that model predictions bear to the type of data used in calibrating the model, and the more similar is the stress regime in which predictions are made to those prevailing at the time of calibration, the greater is the likelihood that parameter uncertainty arising as an outcome of the calibration process will **not** result in a large degree of predictive uncertainty. If parameter uncertainty "doesn't matter" in terms of the model's ability to replicate historical system conditions, then it will quite possibly not matter in terms of its ability to predict future conditions, provided things are not too different in the future. However where conditions are different, and where predictions are of a different type from those used in the calibration process, it is possible that a large degree of predictive uncertainty may be associated with model parameter uncertainty.

Hence parameter uncertainty is, in itself, not a big issue. Parameter uncertainty is important only in so far as it effects predictive uncertainty, and this depends on the prediction. (Unless, of course, the role of the inversion process is actually to infer parameters for their own sakes; in this case the parameters themselves are the predictions. The discussion that follows is directly applicable to this situation when parameters are considered in that light.)

Figure 6.3 illustrates the dependence of a particular model prediction on parameter values for the two parameter system represented in Figure 6.2. As can be seen from Figure 6.3, the prediction of interest increases with both p_1 and p_2 (the values of the two parameters). If it is our desire to find the maximum prediction of this type that is compatible with the fact that model parameters must be constrained such that the model correctly simulates system performance under calibration conditions, then this prediction is seen to correspond to the "critical point" depicted in Figure 6.4. This is the point of maximum model prediction on the $\Phi_{min} + \delta$ contour. The model prediction made with parameter values corresponding to this critical point is the maximum model prediction that is compatible with calibration-imposed constraints on parameter values. In many instances of model usage, this particular prediction is of greater importance than a model prediction made with best-fit parameter values (ie. using parameters corresponding to Φ_{min}).

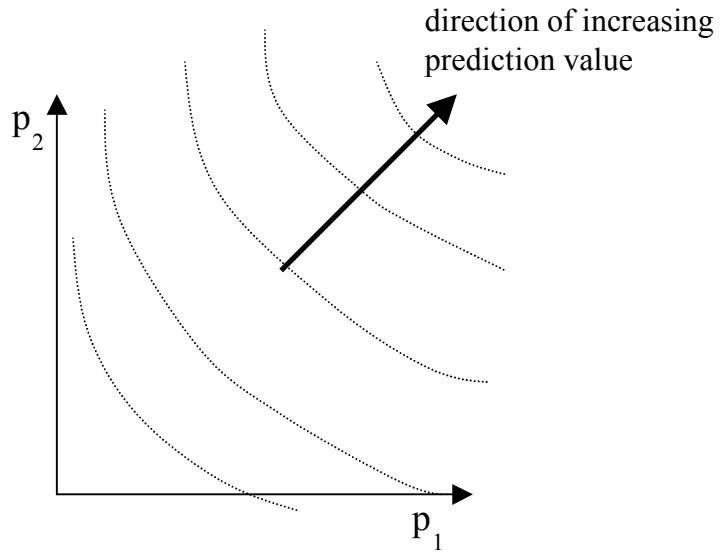


Figure 6.3 Contours of a model prediction in parameter space.

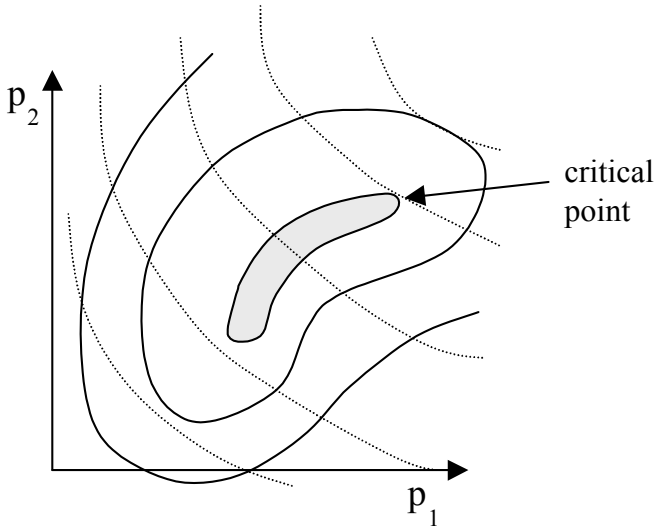


Figure 6.4 The "critical point" in parameter space.

The situation is only slightly more complicated when parameter bounds are imposed. Figure 6.5 shows objective function contours, model prediction contours, and the critical point in this case. The prediction corresponding to parameter values at the critical point is now the worst or best model prediction that it is possible to make after both knowledge and calibration constraints have been imposed.

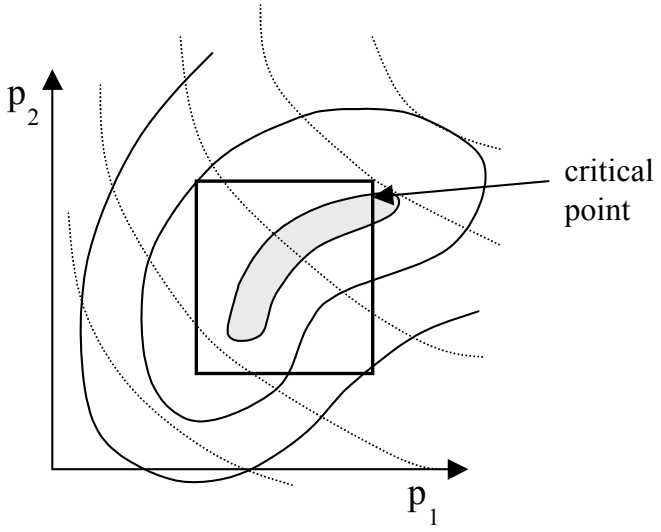


Figure 6.5 The "critical point" with parameter bounds taken into account.

The importance of predictive analysis in model deployment cannot be understated. In most instances of model usage only a single prediction is made. However where a range of predictions is possible, it is poor modelling practice not to provide at least some indication of the extent of this range. The model prediction corresponding to the critical point defines the extent of this predictive range in one direction.

For example, if a rainfall-runoff model has been calibrated in order to make predictions of flood height following future high rainfall events, then the model's prediction of the maximum possible flood height may be of critical importance to formulation of a flood management strategy. Due to the fact that the height of the flood peak may be sensitive to parameters which are highly correlated with each other under the more benign conditions under which the model was calibrated, the possibility of predictive uncertainty is very high; so too is the imperative for predictive analysis.

6.1.4 Dual Calibration

There are two means whereby PEST can be used to establish the degree of uncertainty associated with a particular model prediction. The first is approximate and can be done using PEST under normal parameter estimation conditions. The second involves determination of the actual critical point. The first of these is discussed in the present section while the second is discussed in the next section.

"Dual Calibration" gets its name from the fact that PEST is asked to calibrate a model that is actually comprised of two models. Recall that the "model" run by PEST can actually be a batch file comprised of many executables. A batch file can easily be written such that it first runs the model under calibration conditions and then runs the model under predictive conditions. The dataset for "calibration" of this dual model should be the normal calibration dataset (for which corresponding model-generated numbers are produced by the first component of the dual model, ie. the model run under calibration conditions), plus a single extra "observation" which corresponds to a best or worst case model outcome of a certain type for which the corresponding model-generated number is produced by the second

component of the dual model. PEST is thus asked to calibrate this dual model such that, on the one hand, the model produces numbers which match historical system behaviour, while on the other hand it produces a certain best or worst case prediction. If a parameter set can be found which allows the model to do this, then the worst or best case prediction is compatible with calibration constraints (and knowledge constraints if parameter bounds are imposed). If it cannot, then the hypothesised worst case or best case prediction is not possible.

The trick in implementing the dual calibration technique is the selection of a weight to assign to the single model prediction. In general, the weight applied to this prediction should be such that the contribution to the final objective function by the residual associated with the single prediction is of the same order as the contribution to the objective function by all of the residuals associated with the historical component of the dual model. If this is the case, then obviously the model will not be producing a prediction which is exactly equal to the user-supplied best or worst case prediction. Because of this, the prediction may have to be supplied as a little "worse than worst" or "better than best". A strategy for choosing a suitable weight and predictive value will soon become apparent once the method is implemented on a particular case.

Dual calibration can be used with any nonlinear parameter estimator. However it is particularly easy to implement with PEST. One reason for this is the model-independent nature of PEST which allows it to be used as easily with a composite model encapsulated in a batch file as with a model comprised of a single executable. Another reason lies in the robust nature of the PEST inversion algorithm. Yet another reason lies in the fact that PEST prints out the contribution made to the objective function by different observation groups. Thus if historical observations used with the calibration component of the composite model are assigned to one observation group, and the single model prediction made under future conditions is assigned to another observation group, the user can see at a glance what the "calibration component" of the objective function is, and what the "predictive component" of the objective function is. Together, these add up to the total objective function whose task it is for PEST to minimise during the dual calibration process.

If one of the names of the observation groups supplied to PEST is "predict", and if there is only one observation belonging to that group (this observation can have any name), then 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 prints out the line:-

```
prediction = x
```

where *x* is the model-generated value corresponding to the sole observation belonging to observation group "predict". Obviously, if you are not using PEST to undertake dual calibration, you should avoid use of "predict" as an observation group name.

6.1.5 Predictive Analysis Mode

PEST can run in three different modes – "parameter estimation mode", "regularisation mode" and "predictive analysis mode". The first of these modes is used to find the objective function minimum Φ_{min}; the second is most useful when many parameters require adjustment through the calibration process; see the next chapter. When run in predictive analysis mode, PEST

finds the critical point depicted in Figures 6.4 and 6.5 and determines the model prediction associated with that point. The theory underpinning PEST's calculations when run in predictive analysis mode is presented in Section 2.1.9.

Model setup for predictive analysis is similar to that for dual calibration operations. That is, a composite model is constructed comprised of the model run under calibration conditions followed by the model run under predictive conditions. There can be as many field observations corresponding to the former model component as desired, these being the "calibration observations". However there should be only one output from the predictive model component which is used by PEST as an observation. Furthermore, so PEST can recognise this observation, it should be the sole member of an observation group named "predict". *It is important to note that PEST takes no notice of either the "observed value" of this observation or of the weight assigned to this observation. PEST's job is simply to raise or lower the model output corresponding to this observation, while maintaining the objective function at or below* $\Phi_{min} + \delta$.

Before using PEST to undertake predictive analysis, you should have already calibrated the model. Because the model has been calibrated, you will have determined a parameter set corresponding to the objective function minimum; you will also have determined the objective function minimum itself, ie. Φ_{min}. When run in predictive analysis mode PEST must be supplied with the value of $\Phi_{min} + \delta$, henceforth referred to as Φ_0. PEST then maximises or minimises the model prediction (you tell it which), while ensuring that the objective function (calculated on the basis of calibration observations alone) is as close as possible to Φ_0. See Section 2.1.9 for further details.

PEST's operation in predictive analysis mode has much in common with its operation in parameter estimation mode. 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 parameter estimation process. If they are outside of the Φ_0 contour, PEST automatically works in parameter estimation mode until it is "within reach" of Φ_0, at which stage it modifies its operations to search for the critical point.

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 or three 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 log-transformed, linked to one another, or fixed in predictive analysis mode just as in parameter estimation mode. In fact parameter 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 parameter 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 parameter 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" or "/j" switch just as in parameter estimation mode (provided that the PEST RSTFLE variable was set to "restart" on the previous run). Relative and factor change limits are just as important when PEST is used in predictive analysis mode as when it is used in parameter 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 parameter 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 single observation belonging to the "predict" group (which, obviously, does not figure in the previous PEST calibration run) can be assigned any weight at all; this weight is ignored by PEST when run in predictive analysis mode.

When run in predictive analysis mode more model runs are normally required to achieve solution convergence than are required when PEST is run in parameter estimation mode because it is usually a more difficult matter to find the critical point than it is to find the objective function minimum.

Except for the value of one control variable, the same control file can be used by PEST when run in both parameter estimation and predictive analysis modes (provided the control file has a "predictive analysis" section – see below). Furthermore, because of the functionality attached to the observation group "predict", it is a particularly simple matter to run PEST in predictive analysis mode on a particular problem and then switch to running PEST in parameter estimation mode on the same problem in order to implement a dual calibration exercise.

PEST screen output is slightly different when run in predictive analysis mode from its screen output when run in parameter estimation mode in that the value calculated for the prediction is written to the screen on every 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 file *case.par* where *case* is the filename base of the PEST control file.

As has already been mentioned, 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 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 in objective function values.

6.2 Working with PEST in Predictive Analysis Mode

6.2.1 Structure of the PEST Control File

The PEST control file used for running PEST in predictive analysis mode is shown in Example 6.1 An example of this file is provided in Example 6.2.

```
pcf
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP
NTPLFLE NINSFLE PRECIS DPOINT NUMCOM JACFILE MESSFILE
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM
RELPARMAX FACPARMAX FACORIG
PHIREDSWH
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR
ICOV ICOR IEIG
* parameter groups
PARGPNME INCTYP DERINC DERINCLB FORCEN DERINCMUL DERMTHD
(one such line for each of the NPARGP parameter groups)
* parameter data
PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM
(one such line for each of the NPAR parameters)
PARNME PARTIED
(one such line for each tied parameter)
* observation groups
OBGNME
(one such line for each observation group)
* observation data
OBSNME OBSVAL WEIGHT OBGNME
(one such line for each of the NOBS observations)
* model command line
write the command which PEST must use to run the model
* model input/output
TEMPFLE INFLE
(one such line for each model input file containing parameters)
INSFLE OUTFLE
(one such line for each model output file containing observations)
* prior information
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of the NPRIOR articles of prior information)
* predictive analysis
NPREDMAXMIN
PD0 PD1 PD2
ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH
ABSPREDSWH RELPREDSWH
NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP
```

Example 6.1. Construction details of the PEST control file for use in predictive analysis mode.

```
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
1.0  1.05   2.0
0.00 0.005 1.0 2.0 8
0.00 0.05
4    0.0    0.005   4
```

Example 6.2. Example of a PEST control file for use in predictive analysis mode.

Differences between the PEST control file used by PEST for running in predictive analysis mode, and that used for work in parameter estimation mode are few. The PESTMODE variable on the third line of the PEST control file must be set to "prediction". Also, the PEST control file must contain a "predictive analysis" section which controls PEST's operations in this mode. Note the following:-

- if PEST is run in parameter estimation mode the "predictive analysis" section of the PEST control file is ignored and can, in fact, be omitted;

- if there is no prior information the "prior information" section of the PEST control file should either be omitted or simply left empty.

6.2.2 PEST Variables used for Predictive Analysis

The role of those PEST variables which govern its operation in predictive analysis mode will now be discussed in detail. Although PESTMODE was discussed in Section 4.2.2, specifications for this variable will now be repeated. All other variables discussed below reside in the "predictive analysis" section of the PEST control file and hence pertain only to PEST's operation in predictive analysis mode.

PESTMODE

This is a character variable that appears on the third line of the PEST control file just after the RSTFLE variable. It must be supplied as either "estimation", "prediction" or "regularisation". In the first case PEST will run in parameter estimation mode (its traditional mode of operation); in the second case PEST will run in predictive analysis mode; in the third case PEST will run in regularisation mode (see Chapter 7).

As mentioned above, if PEST is run in predictive analysis mode, then you must ensure that the PEST control file contains a "predictive analysis" section. You must also ensure that there are at least two observation groups, one of which is named "predict", and that the "predict" group has just one observation. In most cases this will correspond to an output of the predictive component of a composite model. The observation can have any name, value and weight; the latter two are ignored by PEST when run in predictive analysis mode. (Naturally you must supply an instruction file to read this extra observation from the pertinent model output file.)

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 $\Phi_{min} + \delta$ (ie. Φ_0). If NPREDMAXMIN is set to 1, PEST will maximise the prediction; if NPREDMAXMIN is set to –1, PEST will minimise the prediction.

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 $\Phi_{min} + \delta$, ie. Φ_0 (see Section 6.1.1). 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.

PD1

The procedure by which PEST calculates the location of the critical point in parameter space is a complex one; see Section 2.1.9. If PEST is asked to maximise (minimise) a certain model prediction while constrained to keep the objective function (calculated on the basis of

calibration observations only) as close as possible to PD0 it will, in the course of its iterative solution process, wander in and out of "allowed parameter space" (ie. the area inside the $\Phi_0 + \delta$ contour of Figures 6.1, 6.2, 6.4 and 6.5).

Because the shape of the PD0 contour can be so complex, it is extremely hard for PEST to find a parameter set which lies exactly on this contour. The value supplied for PD1 (which must be slightly higher than PD0) is a value which PEST will consider as being "close enough" when approaching this contour from the outside. (When approaching it from the inside, any objective function value is "good enough" because values on the inside of the PD0 contour are all less than PD0 and hence all calibrate the model.)

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. This should normally be about 5% higher than PD0. However if you are not undertaking a line search for refinement of the parameter upgrade vector (see below), or if PEST appears to be having difficulties in finding parameter sets which can raise or lower the prediction (whichever is appropriate) while keeping the model calibrated, it may be advisable to raise PD1 to 10% higher than PD0.

PD2

When used in predictive analysis mode, the solution procedure used by PEST is very similar to that used in parameter estimation mode. During each optimisation 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 (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 4.2.2 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 $(\Phi - \Phi_0)$ (ie. 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 Section 2.1.9 of this manual. However it has been found from experience that the critical point of Figures 6.4 and 6.5 can be found more efficiently 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 optimisation 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 between 6 and 8.

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 of Section 2.1.9 alone. Unless there is a good reason to do otherwise, a value of 1.0 is appropriate here. Then 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 1.5 to 2.0 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.

It may seem at first sight that implementation of a line search algorithm may prove very costly in terms of model runs. However experience to date is such as to suggest that inclusion of the line search option may result in a dramatic reduction in overall model runs, as fewer optimisation iterations are required to find the critical point, even though each iteration may individually require more model runs.

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, or two points at first and then three points later. When run in parameter estimation mode, PEST makes the switch between two point and three point derivatives calculation if it fails to lower the objective function by a relative amount equal to PHIREDSWH between successive optimisation 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 three point 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 three point 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

The last four variables are termination criteria.

When PEST is used in parameter estimation mode, the optimisation 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 optimisation 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 optimisation 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 optimisation 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 for PEST's usage in parameter estimation mode.

6.3 An Example

See Section 12.2 for an example of PEST's use in Predictive Analysis Mode.

7. Regularisation

7.1 About Regularisation

7.1.1 General

PEST (and other nonlinear parameter estimation software) sometimes encounters difficulties in minimising the calibration objective function where too many parameters must be simultaneously estimated. Such situations often arise when calibrating models that represent two- and three-dimensional spatial processes, or when using such models to infer the properties of a two- or three-dimensional model domain as part of a data interpretation process. In many cases the user may not wish to observe the conventional wisdom of parameter parsimony for fear of losing important system details. However the use of too many parameters leads to numerical instability and nonuniqueness of parameter estimates.

In an attempt to reduce the number of parameters to a manageable level, a two- or three-dimensional model domain is often subdivided into a small number of zones of assumed parameter constancy; zone boundaries can be chosen on the basis of geological and/or other evidence (where this evidence exists), or inferred from the field data itself. While this methodology works well, there are many problems associated with it. For example, zonation of the model domain may be difficult as it may not be immediately apparent from the observation dataset where property contrasts exist, or whether property transitions are smooth rather than abrupt. In many cases, the modeller may prefer to ask PEST to infer areas of high or low property value itself, rather than attempting to construct a parameter zonation scheme based on an incomplete knowledge of the locations of property discontinuities.

Unfortunately a calibration and/or data interpretation strategy which does not rely entirely on an externally-supplied zonation will nearly always result in the necessity to estimate a large number of parameters. Some of these parameters will invariably be more sensitive to the field data than others; insensitive parameters are not only difficult to estimate themselves, but may hamper PEST's ability to estimate other, more sensitive, parameters (see Section 5.6 for details). Furthermore, parameter correlation is often very strong in highly parameterised systems, this resulting in a high degree of nonuniqueness in parameter estimates. Hence even if a good fit is obtained between model outputs and field data, the parameters giving rise to this fit will probably be just one set out of a virtually infinite number of sets that will also result in a good fit between model outputs and field data.

A related problem in working with highly-parameterised systems is that unless some constraints are imposed on parameter values, or on relationships between parameter values, individual parameter estimates tend to show a high degree of spatial variability, and can even take on extreme values, as PEST tries to use every parameter at its disposal in order to accommodate every nuance of the observation dataset upon which the calibration or data interpretation process is based. In many cases these nuances are better considered as "system noise". Hence an appropriate level of misfit between model outputs and field data can be tolerated on this basis, for it is well known that a parameterisation which attempts to fit all of the "fine detail" of a measurement dataset is often a poor representation of true system properties. Use of an appropriate regularisation procedure in the parameter estimation process

overcomes this problem by allowing the modeller to inject some "sanity" into the process by limiting the degree of spatial variability that optimised parameter values are allowed to show. While this may be done at the cost of obtaining a perfect fit between model outputs and field data, the estimated parameters are often far more believable as result of this misfit. Furthermore, as will be described below, when run in "regularisation mode" the user is able to inform PEST of the maximum cost that he/she is prepared to tolerate in terms of model-to-measurement misfit when implementing a regularisation constraint.

The power of a properly-implemented regularisation procedure is that it allows the modeller access to the benefits of using a large number of parameters (in terms of the spatial resolution required to define the location of important property contrasts), at the same time as it allows him/her access to the benefits of a more sparingly parameterised system (in terms of stability of the inversion process, reduced parameter correlation, and a reduced propensity for estimated parameter values to be wild and unbelievable). It also allows the user to inject his/her judgement into the parameterisation process by selecting a regularisation scheme that reflects his/her current state of knowledge of that system.

7.1.2 Smoothing as a Regularisation Methodology

The most commonly used regularisation methodology is the imposition of a "smoothing constraint" on parameter values. In most cases this is achieved by taking differences between neighbouring pairs of parameter values (or between functions of these parameter values) and requesting that each such difference be zero if possible. This is done by supplying each parameter difference to PEST as an extra "observation", or as an article of prior information, the "observed value" for which is zero in each case. Thus each non-zero parameter difference makes a contribution to the objective function whose task it is for PEST to minimise. Hence in minimising the objective function, PEST seeks to minimise these parameter differences and, in so doing, increases parameter uniformity across the model domain (or of sub-areas within this domain - see below).

Regularisation can be used in conjunction with most methods of defining a parameterisation scheme over a model domain. It can be used in conjunction with zones of assumed parameter constancy, for use of an appropriate regularisation scheme allows the modeller to use more zones than he/she otherwise would. Through enforcing either a single, regional smoothing condition, or a series of more local smoothing conditions over different model sub-areas, the model can be parameterised in such a way that it respects outside knowledge (for example geological knowledge) at the same time as it accommodates smaller scale property variations in order to provide a satisfactory fit between model outputs and field measurements. Similarly, if a "pilot points" parameterisation scheme is used (in which parameter values are estimated for points lying within the model domain, and these parameter values are then spatially interpolated to the cells or elements of the model grid or mesh using kriging or some other spatial interpolation method), the degree of spatial parameter variation between these pilot points, or of various subgroups of them, can be limited through the imposition of appropriate uniformity criteria.

As stated above, regularisation can take place across the entire model domain, or across sub-areas within the domain. In the latter case, if there are boundaries within the model domain where property discontinuities are known to exist, then no parameter differences should be taken across these boundaries for inclusion in the "regularisation observations" used to

enforce the smoothing condition.

Regularisation criteria other than uniform smoothing can also be used. For example individual parameter differences might be weighted according to inter-parameter distance. Or a minimum curvature condition could be imposed; each relationship comprising this regularisation scheme will cite at least 3 parameters, stating that the difference between the "inner" parameter value and the average of at least two "outer" parameter values is zero. Or a regularisation scheme might be comprised of a series of differences between many (or all) of the model's parameters and some preferred value for each of these parameters. Even more sophisticated techniques might be developed; for example a regularisation condition imposed on the calibration of a transient model might be that the same parameter distribution which calibrates the transient model also calibrates a steady state model. There is no limit to the range of possibilities.

The main criteria for a regularisation methodology are that:-

1. it includes a substantial number of relationships between most or all of the parameters involved in the parameterisation process, and

2. it encapsulates some "preferred state" of the system; deviations from this "preferred state" are tolerated only to the extent that they allow the model to provide an acceptable fit to field measurements.

Because regularisation should attempt to impose some "preferred state" on system parameters, and because it should involve as many of the model parameters as possible, it constitutes a mechanism for making insensitive parameters sensitive. Without the use of a suitable regularisation strategy, a parameter pertaining to a part of the model domain far removed from locations where field measurements were made, may have a sensitivity of almost zero. Thus if that parameter is estimated (together with other model parameters) on the basis of field data alone, its estimated value will be subject to a very large margin of uncertainty. However if a regularisation condition is imposed, the value estimated for such an ill-determined parameter will most likely be the "natural value" of the parameter as defined through the regularisation process. Similarly if, when estimated on the basis of field measurements alone, certain parameters are highly correlated with each other, each member of the correlated set could take on an infinite number of values, provided other members of the set take on complementary values. Imposition of a properly-constructed regularisation condition will result in the selection of just one set of values for these correlated parameters, ie. the set of values that is most in harmony with the "natural" state of the system as defined by the regularisation conditions.

7.1.3 Theory

The theoretical underpinnings of the regularisation methodology provided in PEST is presented in Section 2.1.10 of this manual.

7.2 Implementation in PEST

7.2.1 Regularisation Mode

To introduce regularisation into the parameter estimation process in the manner described in Section 2.1.10, PEST must be run in "regularisation mode". This mode of operation is not too different from PEST's traditional parameter estimation mode in that an objective function is minimised, the objective function being defined as the sum of squared weighted differences between observations and corresponding model outputs. However when used in regularisation mode, observations must be subdivided into "measurement observations" and "regularisation observations" so that PEST can calculate the contribution made to the total objective function by each of these. Furthermore, during each optimisation iteration, just after the Jacobian matrix has been filled, and just before parameter upgrade vectors are calculated and tested, PEST calculates a suitable "regularisation weight factor" (ie. μ from equation 2.33) by which all of the weights pertaining to the regularisation observations are multiplied prior to calculation of the parameter upgrade vector. This regularisation weight factor is calculated in such a way as to try to ensure that the measurement objective function after parameter upgrade is equal to Φ_m^l defined in equation 2.31, ie. the "limiting measurement objective function" below which the model is deemed to be calibrated. However, as it is calculated on the basis of a linearity assumption (based on the Jacobian matrix) that is not always a good approximation to reality, measurement objective functions equal (or nearly equal) to Φ_m^l are not normally achieved until late in the parameter estimation process.

7.2.2 The Observation Group "Regul"

Regularisation observations are distinguished from measurement observations through being assigned to a special observation group named "regul". As was discussed in Section 4.2.6, prior information items, as well as observations, should be assigned to observation groups. Thus regularisation observations can be supplied in either the "observation data" or "prior information" sections of the PEST control file, or both. All observations and/or prior information equations which do not belong to the group "regul" are assumed to be part of the "measurement" dataset. The weights used by these measurement observations and/or prior information equations are not varied during the parameter estimation process.

The user must also supply PEST with a value for the limiting measurement objective function Φ_m^l. In some cases, PEST will be run in regularisation mode only after it has been used on the same problem in parameter estimation mode. If this is the case, the user will know the lowest value for Φ_m that can be achieved without any regularisation constraints imposed, and will set Φ_m^l a little higher than this. Alternatively, if no preceding parameter estimation run has been carried out, Φ_m^l can be set at a level that is judged to be appropriate on the basis of assumed measurement standard deviations. If PEST cannot lower the measurement objective function as low as Φ_m^l, then it will simply lower it as far as it can (and will normally calculate a low regularisation weight factor in order to achieve this).

While PEST's operation in regularisation mode is similar in many respects to its operation in parameter estimation mode, there are some important differences. In both modes of operation PEST attempts to lower an objective function; however in regularisation mode the total objective function cannot be compared from iteration to iteration, for the composition of the

objective function changes with the regularisation weight factor μ depicted in equation 2.33. However each of the separate components of the objective function, viz. Φ_r (the regularisation component) and Φ_m (the measurement component) can be compared from iteration to iteration.

Because, when operating in regularisation mode, PEST's intention during each optimisation iteration remains the same as in parameter estimation mode, ie. to lower an objective function, all of the input variables which control its operation in parameter estimation mode are still required for its operation in regularisation mode. Furthermore, they still have the same roles. However, as will be discussed below, a number of new variables are required in the PEST control file to control PEST's operation in regularisation mode.

7.3 Preparing for a PEST Run in Regularisation Mode

7.3.1 The PEST Control File - "Control Data" Section

As was discussed in Section 4.2.2 of this manual, the variable PESTMODE on the third line of the PEST control file must be provided as "estimation", "prediction" or "regularisation"; the last of these options must be supplied for this variable for PEST to run in regularisation mode. If so, there must be a "regularisation" section at the end of the PEST control file (following either the "model input/output" section or, if present, the "prior information" section of the PEST control file).

7.3.2 The PEST Control File - 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, one of which must be named "regul". All observations and/or prior information items belonging to the group "regul" comprise the "regularisation observations"; all observations and/or prior information equations belonging to all other groups comprise the "measurement observations". Weights must be assigned to individual observations and prior information equations within each group in the normal manner (or an observation covariance matrix can be assigned - see Section 4.3). However, as was explained above, weights assigned to the regularisation observations and/or prior information equations are multiplied internally by a "regularisation weight factor" prior to formulation of the total objective function during each optimisation iteration.

7.3.3 Control File - "Regularisation" Section

Example 7.1 shows the format of the "regularisation" section of the PEST control file. This section should be placed at the end of the file.

```
* regularisation
PHIMLIM  PHIMACCEPT FRACPHIM
WFINIT  WFMIN  WFMAX
WFFAC  WFTOL
```

Example 7.1 Format of the "regularisation" section of the PEST control file.

An example of the "regularisation" section of a PEST control file is provided in Example 7.2.

```
* regularisation
135.0  140.0 0.0
1.0e-2  1.0e-6 1.0e6
1.3  1.0e-2
```

Example 7.2 An example of the "regularisation" section of a PEST control file.

The "regularisation" section of the PEST control file must begin with a single line containing the character string "***** regularisation". Then follow three lines, each of which contains a number of variables which control the way in which PEST operates when working in this mode. The role of each of these variables is now discussed.

PHIMLIM

This is Φ_m^l of equation 2.31. That is, it is the upper limit of the measurement objective function (ie. the upper level of model-to-measurement misfit) that is tolerable when trying to minimise the regularisation objective function Φ_r. In some cases a PEST regularisation run will postdate a normal parameter estimation run. If the latter run was successful, it will have informed the user of how low the measurement objective function can be if all parameters are adjusted without reference to any regularisation conditions. Φ_m^l should be set somewhat above this, for the imposition of regularisation constraints will mostly result in a slight diminution of PEST's ability to fit the field data exactly. The user informs PEST of the extent to which he/she will tolerate a less-than-optimal fit between model outputs and field data for the sake of adhering to the "reality check" imposed by the regularisation constraints through the variable PHIMLIM.

PHIMACCEPT

During each optimisation iteration, just after it has linearised 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 μ 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 component of the objective function is equal to PHIMLIM. However, due to the approximate nature of the linearity assumption, PEST may not be able to lower the measurement component of the objective function to PHIMLIM 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 the measurement objective function to an acceptable level, it simply accepts the upgraded parameters, proceeds to the next optimisation iteration and tries again. However if it does succeed in lowering the measurement objective function 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 the regularisation component of the objective function Φ_r while maintaining the measurement component of the objective function Φ_m below this acceptable level. This acceptable level is PHIMACCEPT; it should be set slightly higher than PHIMLIM (ie. 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% to 10% 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 optimisation 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 an 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 restricted somewhat, for there may be some parameter sets for which the measurement objective function 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.

FRACPHIM

PEST ignores the value supplied for FRACPHIM unless it is greater than zero. A value of between zero and 1.0 (but normally less than about 0.3) can be supplied for this variable if you are unsure what value to use for PHIMLIM. See Section 7.3.4 below for a full discussion of this variable.

WFINIT

This is the initial regularisation weight factor. During every optimisation iteration PEST calculates a suitable regularisation weight factor to use during that optimisation iteration using an iterative, numerical solution procedure; its initial value when implementing this procedure for the first optimisation iteration is WFINIT. If there are many adjustable parameters, calculation of the regularisation weight factor for the first optimisation iteration can be very 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.

WFMIN, WFMAX

These are the minimum and maximum permissible values that the regularisation weight factor is allowed to take. If a regularisation scheme is poor, (and does not lend too much stability to an already unstable parameter estimation process), selection of appropriate values for WFMIN and WFMAX may be quite important, for these can prevent PEST from calculating outrageous values for the regularisation weight factor in an attempt to compensate

for inadequacies of the regularisation scheme.

A regularisation scheme should be such that, even if there are no field measurements, it can "almost" result in a unique parameter set all by itself; "almost" here implies that there may still be a degree of nonuniqueness, but that this might only be in relation to a factor by which all parameters can be multiplied and still satisfy the regularisation conditions. However if the regularisation scheme is such as to allow high and untrammelled variability of parameters, PEST can encounter serious difficulties if the inversion problem to which the regularisation scheme refers is already unstable. In the belief that if regularisation observations were to make a more substantial contribution to the objective function than measurement observations, greater stability will ensue in accordance with the aims of introducing regularisation into the optimisation process in the first place, PEST may try to assign greater weights to regularisation observations than it otherwise would when the inversion process runs into difficulties. Where the number of adjustable parameters is high and numerical solution of the weight factor equation is thus very time-consuming, this may place an undue strain on PEST's operation in regularisation mode. If the upper weight factor limit, WFMAX, is set at an appropriate value, the user will be able to detect this aberrant behaviour sooner than he/she otherwise would, and take appropriate action if necessary.

WFMIN and WFMAX values of 10^{-6} and 10^{6} respectively are suitable for most occasions.

WFFAC, WFTOL

When PEST calculates the appropriate regularisation weight factor to use during any optimisation iteration, it uses an iterative procedure which begins at the value of the regularisation weight factor calculated for the previous optimisation iteration; for the first optimisation 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 (ie. Φ_m) of PHIMLIM (ie. 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 optimisation 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. WFTOL is best set at somewhere between 10^{-3} and 10^{-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 10^{-2} may prove suitable.

7.3.4 The Control Variable FRACPHIM

As was mentioned above, a non-zero value can be supplied for FRACPHIM if you would like to use PEST in regularisation mode, but you are unsure of what value to use for PHIMLIM.

If FRACPHIM is provided with a value of zero or less (or if this variable is absent from the PEST control file), then PEST's action when working in regularisation mode will be exactly the same as that already described. 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 will calculate a new value for PHIMLIM at the beginning of each optimisation iteration. This value will be calculated as the current value of the measurement objective function times FRACPHIM. Thus PEST will always "aim for" a measurement objective function that is lower than the current one. This allows it to lower the measurement objective function as the parameter estimation process progresses while, at the same time, making use of the numerically stabilising effects of regularisation.

The following aspects of the use of the FRACPHIM variable should be carefully noted:-

1. If FRACPHIM is supplied as 0.0, or as less than 0.0, PEST will assume that no value has been supplied for FRACPHIM at all. The value of PHIMLIM used in every optimisation iteration will thus be that supplied in the PEST control file.

2. If FRACPHIM is supplied with a value of 1.0 or greater, PEST will cease execution with an error message.

3. For other values of FRACPHIM, PEST will adjust the value of PHIMLIM at each optimisation iteration by multiplying the current value of the measurement objective function by FRACPHIM. *However it will lower PHIMLIM no further than the value for this variable supplied in the PEST control file.*

4. Optimal values for FRACPHIM are normally in the range 0.1 to 0.3.

5. As well as adjusting the value of PHIMLIM during every optimisation iteration, PEST also adjusts the value of PHIMACCEPT. This adjustment is made such that, during every optimisation iteration, the ratio of PHIMACCEPT to PHIMLIM is the same as that supplied in the PEST control file. Normally this ratio should be no greater than 1.1.

7.4 Working with PEST in Regularisation Mode

7.4.1 Run-Time Information

As it runs, PEST records information to both the screen and to its run record file. When PEST is run in parameter estimation mode, the principal items of interest during each optimisation iteration are the value of the objective function at the beginning of the iteration, and new values of the objective function which are calculated 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 (μ in equation 2.33) is different from iteration to iteration, the objective function calculated during one optimisation iteration is not directly comparable with that calculated by PEST during the previous optimisation iteration. However the measurement and regularisation objective functions are comparable from optimisation

iteration to optimisation iteration.

Example 7.3 shows part of a run record file produced by PEST when operating in regularisation mode.

```
OPTIMISATION ITERATION NO.        :    5
   Model calls so far             :   50
   Current regularisation weight factor                  :  0.16403
   Current value of measurement objective function       :   2.8654
   Current value of regularisation objective function    :   2.2663

   Re-calculated regularisation weight factor            :  0.32052
   Starting objective function for this itn. (ie. phi)   :   3.0982


       Lambda =  3.90625E-02 ----->
          Phi =   2.0301      (  0.655 of starting phi)
    Meas. fn. =   1.6652
   Regul. fn. =   3.5520

       Lambda =  1.95313E-02 ----->
          Phi =   2.5257      (  0.815 of starting phi)
    Meas. fn. =   2.0957
   Regul. fn. =   4.1855

       Lambda =  7.81250E-02 ----->
          Phi =   2.2408      (  0.723 of starting phi)
    Meas. fn. =   1.9340
   Regul. fn. =   2.9866

   No more lambdas: phi rising
            .
            .
   Maximum   factor change:  1.650     ["ro9"]
   Maximum relative change: 0.4643     ["ro8"]


OPTIMISATION ITERATION NO.        :    6
           etc
```

Example 7.3 Extract from a PEST run record file.

PEST begins each optimisation iteration by recording the current value of the regularisation weight factor (as calculated during the previous optimisation iteration) and 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 optimisation iteration to optimisation iteration.)* Of course, no weight factor is used in calculation of 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 for the regularisation weight factor for the current iteration. Once it has calculated the regularisation weight factor,

it can calculate an objective function (ie. "phi") for the current optimisation iteration using equation 2.33. 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. As is apparent from equation 2.33, by lowering the objective function PEST will simultaneously lower 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 parameter estimation mode. 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 - see equation 2.33.*

7.4.2 Composite Parameter Sensitivities

As described in Section 5.3.2 of this manual, in the course of its execution PEST records the composite sensitivities of all adjustable parameters to 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 optimisation iteration to optimisation 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 optimisation iteration to optimisation iteration.

7.4.3 Post-Run Information

At the end of the parameter estimation process PEST records information to its run record file, to its residuals file, to its parameter value file, and to its observation sensitivity file.

Information recorded to the run record file is similar to that recorded to this file when PEST operates in parameter estimation mode. Included in this information are parameter uncertainties and the parameter covariance and associated matrices. Regularisation weights are multiplied by the optimised regularisation weight factor prior to computation of these matrices. Caution should thus be exercised in interpreting the information contained in these matrices. Regularisation information serves the very useful purpose of imposing a "reality check" on an overparameterised calibration problem, and of adding numerical stability to the solution of that problem. However it does not constitute part of the measurement dataset, and hence should not really be used in the calculation of the uncertainty associated with each parameter that is estimated on the basis of that dataset.

Like the run record file produced by PEST when run in parameter 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 are multiplied by the optimised regularisation weight factor, ie.

by the regularisation weight factor used in the calculation of optimised parameter values. (Note that these optimised parameter values are also listed on the run record file and, of course, in the parameter value file).

A similar consideration applies to information written to the residuals file at the end of the optimisation process. That is, where weights are used in the calculation of any quantities pertaining to regularisation observations listed in this file, the regularisation weights supplied by the user are multiplied by the optimised regularisation weight factor. Note also that "measurement standard deviations" and "natural weights" are not calculated for regularisation residuals, as these have no meaning.

Information recorded on the observation sensitivity file produced by PEST at the end of its run is also weight-dependent; see Section 5.3.3 for details. In this case, just as in the cases discussed already, weights used for regularisation observations are equal to user-supplied weights multiplied by the optimal regularisation weight factor.

7.5 Other Considerations Related to Regularisation

7.5.1 Using PEST in Two Different Modes

As is explained in Section 7.3, a PEST control file suitable for use by PEST in regularisation mode differs from a PEST control file suitable for use by PEST in parameter estimation mode, in three ways, viz:

1. the PESTMODE variable must be set to "regularisation" rather than "estimation",
2. an observation group named "regul" must be present, and
3. a "regularisation" section must be present at the end of the PEST control file.

Once a PEST control file has been built for PEST usage in regularisation mode, it is a simple matter to use PEST in parameter estimation mode on the basis of the same input dataset. All that needs to be done is to change PESTMODE from "regularisation" to "estimation". PEST will then run happily in parameter estimation mode, ignoring the redundant "regularisation" section at the end of the PEST control file. It will treat the observation group "regul" just like any other group, and members of this group just like any other observations. However, if you wish to dispense with this regularisation information altogether, assign all members of this group a weight of zero.

As was mentioned above, parameter uncertainties and other statistics recorded at the end of the run record file after PEST was run in regularisation mode are calculated on the basis of user-supplied weights for members of the observation group "regul" multiplied by the optimised regularisation weight factor. However if you have just undertaken a regularisation run and you would like to calculate parameter uncertainties (and/or the parameter covariance matrix and other quantities derived from this matrix) on the basis of the measurement dataset alone, you should undertake the following steps.

1. After the regularisation run, create a new PEST control file with optimised parameter values substituted for initial parameter values, using the PARREP utility program

supplied with PEST.

2. Set all regularisation weights to zero.

3. Alter PESTMODE on the new PEST control file to "estimation".

4. Adjust NOPTMAX to -1. As explained in Section 4.2.2 of this manual, when PEST is run with NOPTMAX set to -1, it undertakes enough model runs to calculate the Jacobian matrix, and then terminates execution with a full statistical printout at the end of its run record file.

7.5.2 Initial Parameter Values

When running PEST in regularisation mode, you can supply initial parameter values that are far from optimal, or you can supply optimised parameter values from a previous unregularised PEST run if you wish. In the latter case, the initial measurement objective function Φ_m may actually be less than Φ_m^l.

In most cases it makes no difference to PEST what the starting parameter values are; if they are a long way from optimal, PEST will reduce them to near-optimal before the regularisation mechanism begins to have a strong effect on the optimisation process. However in difficult cases, it has been found from experience that it is sometimes better to start PEST from a previously-optimised set of parameter values. If this is done, the measurement objective function Φ_m will approach PHIMLIM (ie. Φ_m^l) "from the inside" rather than "from the outside". Sometimes this appears to be a more stable procedure. Where run times are long it may also prove to be a faster procedure, as PEST can normally minimise the regularisation objective function, subject to the PHIMLIM constraint, using a fewer number of model runs when starting from an optimal or near-optimal set of parameter values than when starting from parameter values that are far from optimal.

7.6 Two Examples of Regularisation

7.6.1 A Layered Half-Space

Electrical soundings are often undertaken by geophysicists in order to infer the variation of electrical resistivity with depth in the ground. A sounding is carried out by passing electrical current through the ground between two current electrodes placed at varying distances apart from each other and measuring the voltage gradient at the surface induced by this current flow. Interpretation of data gathered in this way is based on the premise that the earth can be simulated as a layered half-space. A model can be used to simulate the electrical response of this half space to the currents that flow through it for different current electrode separations. PEST can then be used in conjunction with this model to infer the electrical properties of the half-space from the sounding data.

A traditional method of inferring half-space properties is to assume that the earth under the measurement site is comprised of a small number of layers, each of uniform resistivity. PEST can then be asked to estimate the resistivity and thickness of each of these layers from the surficial voltage measurements. An alternative method is to assume that there are a large

number of geoelectric layers, and that the boundaries between these layers are situated at logarithmically increasing depths below the surface. PEST can then be asked to estimate the resistivities of these layers. However because of the large number of layers (and hence parameters to be estimated), the level of parameter correlation (and hence of nonuniqueness) will be high. Thus a regularisation constraint must be enforced to stabilise the problem. This can take the form of a series of differences between the resistivities of successive model layers, the "observed value" for each such difference being zero.

Assume that the names provided to PEST for the resistivity of each model layer are "ro1", "ro2", etc. Assume also that these parameters are log-transformed during the parameter estimation process. If regularisation observations are added to the "prior information" section of the PEST control file, this section will look something like Example 7.4. (It is assumed that the subsurface has been subdivided into 15 different layers.)

```
* prior information
pi1 1.0 * log(ro1) - 1.0 * log(ro2) = 0.0 1.0 regul
pi2 1.0 * log(ro3) - 1.0 * log(ro2) = 0.0 1.0 regul
      .
      .
pi15 1.0 * log(ro15) - 1.0 * log(ro1) = 0.0 1.0 regul
```

Example 7.4 Regularisation observations contained in the "prior information" section of a PEST control file.

Note the following points.

- The "observed" value of each parameter difference is 0.0.
- Each element of prior information pertains to the logarithm of the respective parameters rather than to the parameters themselves because the parameters are log-transformed in the parameter estimation process. If it is the user's desire that differences be formed between the parameters themselves rather than between the parameter logarithms, then the differences could be calculated by the model, and an "observed value" of 0.0 for each such observation supplied in the "observation data" section of the PEST control file.
- All of the items of prior information involved in the regularisation process have been assigned to the observation group "regul".
- In the present instance, all of the regularisation observations have been assigned the same weight. (These weights are multiplied internally by the regularisation weight factor before being used by PEST to calculate a new set of parameter values.) If it is the user's desire that PEST try harder to enforce uniformity in some areas rather than others, then this could be requested by adopting a different weights assignment strategy.
- Each parameter difference comprising the set of regularisation observations is formed between a certain layer and the layer underneath it. However for the bottommost layer, the difference is formed between that layer and the topmost layer.

Experience has demonstrated that a regularisation scheme such as that depicted in Example 7.4 works very well. However if an attempt is made to estimate layer thicknesses as well as resistivities, then the scheme breaks down. If layer thicknesses are estimated as well as resistivities, then the regularisation observations on their own do not provide a sufficiently strong constraint on parameter values to constitute a suitable regularisation scheme. The reason for this is obvious when it is realised that if all regularisation conditions were exactly obeyed and the subsurface was uniform, layer thicknesses would be completely indeterminate, for layer thicknesses have no meaning in a uniform half-space.

7.6.2 A Heterogeneous Aquifer

Under steady-state conditions, the flow of ground water through the subsurface is mathematically described by Darcy's law and constrained by the conservation of mass. These two laws can be combined into a single partial differential equation which can be used to calculate the distribution of hydraulic heads throughout a study area for different dispositions of sources and sinks of water, and for different system boundary conditions. Where the shape of the model domain is complex, and/or the hydraulic conductivity of the porous medium through which the water flows is nonuniform, this equation must be solved numerically using a technique such as the finite difference, finite element or analytical element method.

In most ground water modelling applications, the hydraulic conductivity at different locations within the model domain is only poorly known. However if the strengths of the various sources and sinks of water and the characteristics of the pertinent boundary conditions affecting the system are known, then hydraulic conductivities in different parts of the domain can be inferred from borehole water level measurements using PEST (assuming that the coverage of observation bores is good enough). However before doing this, the user must decide on an appropriate parameterisation strategy for the model domain.

A common strategy is to subdivide the model domain into zones of assumed parameter constancy based on geological or other information. Unfortunately, such information is often absent or unreliable. Furthermore, there can be a considerable degree of variation of hydraulic conductivity within each geological unit as a result of many factors including lithological heterogeneity, differential weathering, structural features created during tectonic events, etc.

Thus in many instances it is necessary to consider a more complex parameterisation scheme. A simple but effective scheme is to divide the model domain into a large number of "parameter blocks" (or rectangles) arranged in rows and columns in a grid-like structure, and to then estimate the hydraulic conductivity within each such block. Due to the large number of parameters requiring estimation, a suitable regularisation scheme is essential. One such scheme is to request that differences between block hydraulic conductivities (or their logarithms) in both the row and column directions (assuming that the blocks are arranged in a grid pattern) be reduced to a minimum. The regularisation information can be supplied as prior information, or the parameter differences comprising this information can be calculated by the model. In the latter case, they should be provided to PEST as a series of observations. Hence whenever the model is run, it provides not just the model-generated counterparts to the heads measured at boreholes, but also a series of hydraulic conductivity differences based on the current set of parameters used by the model. If the pertinent regularisation observations are named "d1", "d2", etc, part of the "observation data" section of the PEST control file may

appear as in Example 7.5.

```
o1 0.0 1.0 regul
o2 0.0 1.0 regul
o3 0.0 1.0 regul
etc
```

Example 7.5 Regularisation observations contained within the "observation data" section of a PEST control file.

The following points should be noted.

- The "observed value" of each regularisation observation is zero.
- All regularisation observations belong to the observation group "regul".
- Each regularisation observation should pertain to a single difference between a particular parameter and its neighbour in either the row or column direction.
- For each parameter block, differences should be taken in both the row and column directions (resulting in two regularisation observations). However where a geological boundary occurs between two neighbouring blocks, the difference between parameter values on either side of that boundary should be omitted from the regularisation dataset. In this way the regularisation scheme informs PEST that the preferred locations of hydraulic property contrasts are at recognised geological boundaries. However heterogeneity elsewhere will be accommodated if required.

Other parameterisation schemes, together with appropriate regularisation methodologies, can be used. For example, a "pilot points" methodology is very attractive. Using this technique, PEST is asked to assign hydraulic conductivities to discrete points within the model domain. The hydraulic conductivity at each cell or node of the numerical ground water model is then calculated from the hydraulic conductivities assigned to these pilot points using a spatial interpolation algorithm such as kriging. If appropriate, the interpolation algorithm can be tailored to the geology such that different subsets of pilot points are used as a basis for spatial interpolation within different mapped geological units. For each subset of pilot points a series of differences can be formed between the parameter values assigned to these points in order to create a regularisation scheme not unlike that described above. A Delauney triangulation of the model domain based on pilot point locations can be used to define the set of neighbours to each such point; a parameter difference will then be taken for each neighbouring pair of points. The weight used for each regularisation observation could be independent of the distance between the points, or could be defined as a function of this distance. Where a geological boundary passes between two points, then either the difference would not be taken, or the weight assigned to the pertinent regularisation observation would be zero.

A suite of utility software that implements the use of pilot points for spatial parameter definition in conjunction with the United States Geological Survey ground water flow model MODFLOW is available through the "Ground Water Data Utilities" supplied with PEST.

8. Model-Calculated Derivatives

8.1 General

Accuracy in the calculation of model outputs with respect to adjustable parameters is essential for good PEST performance, especially when working with highly parameterised systems. As well as accuracy, *efficiency* of derivatives calculation is also important, for run-times can be very long when PEST is used with such systems. To enhance its use in conjunction with a complex modelling system, PEST offers the user increased flexibility in derivatives calculation over that which is available through finite differences alone.

8.2 Externally-Supplied Derivatives

8.2.1 The External Derivatives File

Some models are able to calculated 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 itself 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 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". (This is slightly different from the Jacobian matrix in that the latter matrix takes account of whether parameters are log-transformed or not during the inversion process.) 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. Example 8.1 shows an example of an external derivatives file expected by PEST.

```
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
```

Example 8.1. An external derivatives file.

The first line of an external derivatives file contains two integers listing the number of parameters and number of observations represented in the derivatives file. These correspond to the number of columns and the number of rows respectively in the derivatives matrix. They must also agree exactly with the values of the PEST variables NPAR and NOBS in the PEST control file. The derivative matrix is listed next in the file.

Some further aspects of this file are now discussed in detail. These should be noted carefully, for if the external derivatives file does not meet PEST's specifications it will not be read by PEST, or (what is worse) may be read incorrectly by PEST.

8.2.2 File Management

As is further discussed below, it is not necessary that the model supplies an external derivatives file if you do not want it to. However if you notify PEST that the model will supply such a file, then you must also inform PEST of the name of the external derivatives file and of the command which PEST must use to run the model in such a way that it writes this file. Just before issuing this command (it is issued once every optimisation iteration) 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 the user 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.

8.2.3 File Format

The external derivatives file must be an ASCII (ie. text) file in which numbers are separated by spaces, tabs or a comma (it is read by PEST using FORTRAN free field input). As mentioned above, it must be headed by a line citing the number of columns and rows comprising the matrix. The matrix itself must have NOBS rows and NPAR columns, 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.

8.2.4 Derivatives Type

As is documented elsewhere 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.

8.2.5 Use of Derivatives Information

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, the user is able to indicate to PEST the parameters for which derivatives information is supplied externally, and those for which derivatives must be computed by PEST using finite differences.

Even greater complexity 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 purposes 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 and hence 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 could not 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.

8.2.6 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.

8.2.7 Name of the Derivatives File

The user must inform PEST of the name of the external derivatives file through the PEST control file for the current case (see Section 8.5). The external derivatives file can have any legal name except for the following names which are already used by PEST. If the filename root of the current project is *case*, the names to be avoided are:- *case.hld, case.jac, case.jco, case.jst, case.par, case.pst, case.res, case.rmf, case.rmr, case.rst, case.rsr, case.sen*, *case.mtt* and *case.seo*. Other names which must be avoided are *pest.hlt, pest.mmf* and *pest.stp.*

8.2.8 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.

8.2.9 Parallel PEST

PEST cannot receive derivatives through an external file if it is being run as Parallel PEST.

8.3 Sending a Message to the Model

PEST has the ability to send a small "message" to the model prior to running it. This is useful if some aspect of the model's deployment depends on whether it is being run to test parameter upgrades, to calculate derivatives by forward or central differences, or to fill an external derivatives file. 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 Example 8.2

```
derivative_increment
     -2
      4      20
     hcond1       5.005787             1
     hcond2       9.850230             0
     stor1       -5.660591            -2
     stor2        8.257257        -10000
```

Example 8.2. 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, or as the first model run of the PEST optimisation process.

derivative_increment

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.

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 parameter estimation 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 contain up to12 characters. 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 -10000 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, the message file is written to each slave working directory just before the pertinent model run is

initiated by the slave.

8.4 Multiple Command Lines

As has already been discussed, 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. It can do this by reading the PEST-to-model message file.)

With PEST it is possible to use 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 (maybe twice) with the value of the parameter incrementally varied. Thus a different command can now be used to run the model for each such incrementally-varied parameter.

Use of a variable command-line strategy may allow a reduction in overall PEST run time to be achieved 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 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 optimisation 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.

8.5 External Derivatives and the PEST Control File

8.5.1 General

Formatting of the PEST control file must be slightly expanded from that discussed in Section 4.2 to accommodate the use of external derivatives and to control PEST-to-model messaging. Pertinent variables in the PEST control file which govern this aspect of PEST's functionality are now discussed.

8.5.2 "Control Data" Section

As is explained in Section 4.2, the fifth line of the PEST control file begins with the PEST control variables NTPLFLE, NINSFLE, PRECIS and DPOINT. The three variables situated to the end of this line (for which values of 1, 0 and 0 were suggested in Section 4.2) are named NUMCOM, JACFILE and MESSFILE (in that order). (For the sake of backwards

compatibility with older versions of PEST, these variables may be omitted from this line. However it is important to note that either these three variables must be completely absent from the fifth line of the PEST control file, or all of them must be present.)

The roles of these variables are now discussed.

NUMCOM

This 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). For previous versions of PEST, only one command line could be used to run the model. However for versions of PEST from 5.0 onwards, different commands can be used to run the model, depending on the purpose of the model run.

Note that when counting the number of available model command lines when assigning a value to NUMCOM, the command that is used to run the model in order to fill the external derivatives file 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 (which will most often be the case), then NUMCOM should be supplied with a value of 1.

JACFILE

Provide this integer variable with a value of 1 if a special model run is to be undertaken during each optimisation iteration for the purpose of external derivatives calculation. Otherwise provide JACFILE with a value of 0.

Note that if JACFILE is provided with a value of 1 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.

8.5.3 "Parameter Data" Section

As is described in Section 4.2.4, 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 was discussed above, 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 will 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 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, the user should take particular care to provide all elements 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.

8.5.4 "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 Example 8.3, while an example is provided in Example 8.4.

```
* derivatives command line
command to run the model
EXTDERFLE
```

Example 8.3. Structure of the "derivatives command line" section of a PEST control file.

```
* derivatives command line
model_d.bat
derivs.dat
```

Example 8.4. 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, ie. 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.

8.5.5 "Model Command Line" Section

In previous versions of PEST the "model command line" section of the PEST control file contained only a single line, this being comprised of the command that PEST must use to run the model. Indeed if NUMCOM in the "control data" section of the PEST control file is set to 1, then the contents of this section are the same. However if NUMCOM is set to *n*, 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 optimisation 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*.

8.6 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* subdirectory of the PEST directory after installation. See Chapter 12 for a more fully discussed example of the use of PEST and its utilities. In that example derivatives are calculated using finite differences.

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:-

$$y = ax^3 + bx^2 + cx + d \qquad (8.1)$$

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", ie. 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:-

$$p = a + 2b + 3c + 4d \qquad (8.2)$$

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 (ie. a, b, c and d) at each value of x. This is stored internally in the JACOB matrix 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 *parameter estimation* mode, while in the other (*polyp.pst*) it is asked to run in *predictive analysis* mode. In the former case the model "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 fifth line 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* - ie. 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 RELPARMA X and FACPARMAX in Section 4.2.2.)

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 *parameter estimation* mode for a while, concentrating on lowering the objective function until it "sniffs" the "critical point" - ie. 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.

9. Parallel PEST

9.1 Introduction

9.1.1 General

In the course of optimising parameters for a model or of undertaking predictive analysis, PEST runs the model many times. Some model runs are made in order to test a new set of parameters. Others are made with certain parameters temporarily incremented as part of the process of calculating the Jacobian matrix, ie. the matrix of derivatives of observations with respect to parameters (unless derivatives are supplied to PEST directly by the model in accordance with PEST's external derivatives functionality). In calculating the Jacobian matrix, PEST needs to run the model at least as many times as there are adjustable parameters (and up to twice this number if derivatives for some of the adjustable parameters are calculated using central differences). In most cases by far the bulk of PEST's run time is consumed in running the model. It follows that any time savings that are made in carrying out these model runs will result in dramatic enhancements to overall PEST performance.

Parallel PEST can achieve a high degree of performance enhancement by carrying out model runs in parallel. If installed on a machine that is part of a local area network, Parallel PEST can carry out model runs on the different machines which make up the network (including the machine which PEST itself is running on). If model run times are large and the number of parameters is greater than four or five, overall PEST run times can be reduced by a factor almost equal to the number of machines over which Parallel PEST is able to distribute model runs.

As well as allowing a user to distribute model runs across a network, Parallel PEST can also manage simultaneous model runs on a single machine. This can realise significant increases in PEST efficiency when carrying out parameter optimisation or predictive analysis on a multi-processor computer by keeping all processors simultaneously busy carrying out model runs.

The optimisation (including regularisation) and predictive analysis algorithms used by Parallel PEST are no different from those used by the normal PEST. Preparation of template files, instruction files and the PEST control file is identical in Parallel PEST to that of the normal PEST. However use of Parallel PEST requires that one extra file be prepared prior to undertaking an optimisation run, viz. a "run management file". This file informs Parallel PEST of the machines to which it has access, of the names of the model input and output files residing on those machines, and of the name of a subdirectory it can use on each of these machines to communicate with a "slave" which carries out model runs on request; see below.

9.1.2 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 independent, ie. 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 as part of the process of filling the

Jacobian matrix makes this process easily parallelised. Under these circumstances Parallel PEST simply distributes model runs to different machines or processors as they become available, and processes the outcomes of these runs as they are finished.

9.1.3 Parallelisation of the Marquardt Lambda Testing Process

Unlike the Jacobian calculation process, the lambda search process (see Section 2.1.7) is difficult to parallelise. This is because, with the exception of the first two model runs undertaken as part of the lambda search procedure during each optimisation iteration, the Marquardt lambda used at subsequent stages of this procedure is dependent on the results of model runs undertaken on the basis of previous lambda values. Hence it is necessary for PEST to wait until the results of a previous model run have been evaluated before it can undertake a further model run on the basis of a new parameter set calculated using a new Marquardt lambda.

However while the lambda search process is not immediately amenable to parallelisation, it is not impossible to accelerate this process somewhat through "partial parallelisation", thanks to the fact that lambda values used by PEST in this search are all related to each other by multiples of LAMFAC. Thus if PEST is run as Parallel PEST, and if it has access to a number of processors, it can undertake simultaneous model runs across these different processors using parameters calculated on the basis of a series of lambda values related to each other by factors of LAMFAC. To some extent, PEST must "guess" which lambdas to use for these parallelised model runs. If it turns out that some of these lambdas are actually not required, then nothing will have been lost because the respective processors' contributions to the lambda search procedure would have also been zero if they were undertaking no model runs at all.

While Parallel PEST allows such a "partial parallelisation" of the lambda search to be undertaken, parallel lambda runs will **not** be undertaken:-

- if at least one parameter is frozen at its upper or lower bound,
- if PEST is running in predictive analysis mode and a line search is undertaken as part of the predictive analysis process,
- if the model run time for the fastest processor involved in the parallelisation process is less than 1.8 times the model run time for the second fastest processor (see below),
- if only one slave is currently available for the undertaking of model runs (this may happen if all but one slave is currently completing redundant model runs arising out of the previous parallel Jacobian calculation process),
- the user indicates to PEST that only the Jacobian matrix calculation process is to be parallelised (see below).

The user should take particular note of the first of the above exceptions to PEST's ability to undertake a partial parallelisation of the lambda search. The reason for this exception is that while PEST may be able to "guess" the values of future Marquardt lambdas to be used in the lambda search procedure with a high probability of success, it has far more difficulty in

predicting whether a parameter is to be frozen at its upper or lower bound, and the order in which parameters are to be frozen if more than one of them are at their limits. (The fact that a parameter is at its upper or lower bound is no guarantee that it will be frozen, for it may need to move back into adjustable parameter space from that bound as different lambdas are tested).

A continuation of the discussion on how Parallel PEST undertakes "partial parallelisation" of the lambda testing procedure will be presented in Section 9.2.6 below after a discussion of how Parallel PEST actually works.

9.1.4 A Warning

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.

9.1.5 Installing Parallel PEST

The command-line version of the Parallel PEST executable, *ppest.exe* is automatically installed when you install PEST on your machine.

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 optimisation process. To do this, copy *pslave.exe* (also provided with PEST) to an appropriate subdirectory on each such machine. This subdirectory can be the model working directory on that machine if desired; if not, it should be a directory whose name is cited in the PATH environment variable on that machine. Alternatively, if each slave has access to the PEST directory on the "master" machine, PSLAVE can be loaded from that directory each time it is run on each slave machine. This is most easily accomplished if the PEST directory on the master computer, as seen by each slave computer, is cited in the latter's PATH variable.

9.2. How Parallel PEST Works

9.2.1 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. Access to files on other machines is achieved through the use of modern network software. 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. The fact that model input files are written and model output files are read by PEST across a network underlines the point made above that the potential reduction in overall PEST run time that can be achieved by undertaking model runs in parallel will only be realised if model run times are large compared with the delays that may be incurred in writing and reading these files across a network.

9.2.2 The 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 subdirectories, 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 on a particular machine it signals the slave running 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 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 subdirectory 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.

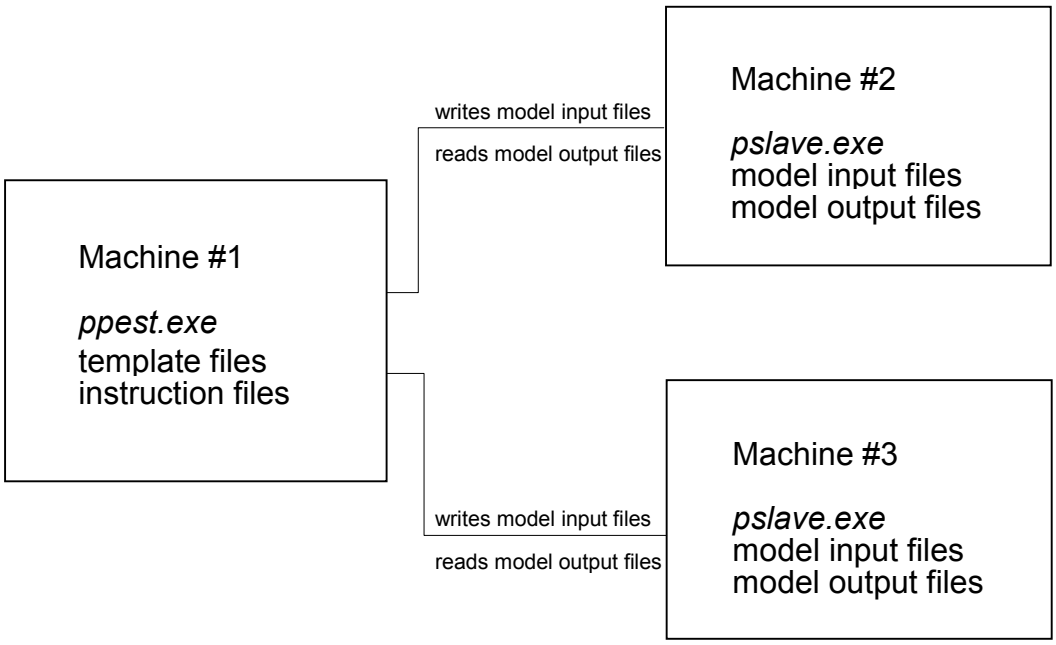


Figure 9.1 Relationship between PEST, PSLAVE and the model.

Figure 9.1 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 (ie. machine #1 in Figure 9.1) to also be a slave machine to avoid wastage of system resources.

9.2.3 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 two incidences of PSLAVE are being run on the same machine in order to gain access to two different 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 optimisation process, the model, as run by that slave, reads its input file(s) and writes its output file(s) to a different subdirectory (or subdirectories) from the model as run by any of the other slaves; this will probably occur naturally when slaves reside on different machines. 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 all input files for one particular model are housed in a single subdirectory; also

all model output files are written to this same subdirectory. In this simple case, preparation for model runs on different machines across a network consists in simply copying the entire model working directory from the master machine to an appropriate directory 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.

9.2.4 Communications between Parallel PEST and its Slaves

Parallel PEST must communicate with each of its slaves many times during the course of the optimisation 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 directory 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 directory from which its execution was initiated. PEST however must be informed of the names of these various PSLAVE working directories (most of which will reside on other machines) through its run management file. Note that there is no reason why a PSLAVE working directory should not also be a model working directory on any slave machine, as long as filename conflicts are avoided. In fact such an arrangement, being simpler, mitigates against making mistakes. Table 9.1 shows the names of the signal files used by PEST and PSLAVE to communicate with each other.

If Parallel PEST carries out two simultaneous model runs on the one machine (for example to exploit that machine's dual processors), then a separate PSLAVE working directory must be commissioned for each separate PSLAVE running on that machine. Once again, these directories may also be the working directories for each of the two 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 files(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 file(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 directories.

Table 9.1 Files used by PEST and PSLAVE to communicate with each other.

Where Parallel PEST is used to run the model on different machines across a network, it is likely that one such slave machine will be the machine that PEST is also running on. This is because PEST 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. Hence the running of PEST leaves adequate system resources available for the running of the model on the same machine. For such a case running PSLAVE from the model working directory, thus designating it as the PSLAVE working directory is, again, entirely appropriate. This directory can also be that from which PEST is run and may thus hold PEST control, template and instruction files. See Section 9.3.8 for a further discussion of this topic.

9.2.5 The Parallel PEST Run Management File

The purpose of the Parallel PEST run management file is to inform PEST of the working directory of each slave (as seen through the network from the machine on which PEST 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 calib
```

then PEST will look to file *calib.pst* to read its control data (ie. *calib.pst* is the PEST control file for the current case) and file *calib.rmf* to read data pertaining to the distribution of model runs across the network.

Example 9.1 shows the structure of a run management file while Example 9.2 shows an example of such a file for the case where there are three slaves.

```
prf
NSLAVE IFLETYP WAIT PARLAM
SLAVNAME SLAVDIR
(once for each slave)
(RUNTIME(I), I=1,NSLAVE)
Any lines after this point are required only if IFLETYP is nonzero; the
following group of lines is to be 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)
```

Example 9.1 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
```

Example 9.2 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, the first of which is the number of slaves, NSLAVE, involved in the current Parallel PEST run. The second item on this line is the value of the variable IFLETYP which must be 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 Example 9.2). However if the names of all corresponding input and output files are the same across all slaves, being identical to the names provided (without subdirectory name) in the PEST control file, and if each set of model input and output files lies within the PSLAVE working directory 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; Example 9.3 shows such a file. In this case PEST automatically affixes the working directory of each slave to the front of each model

input and output file as named in the PEST control file.

The third item on the second line of the PEST run management file is the value for the 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 optimisation process until this question is answered. It would obviously be unfortunate if the 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.

If 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 is further discussed in Section 9.2.6 below.

Lines 3 to NSLAVE+2 (ie. 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 as allocated by the network manager.

The second item on each of these lines (ie. SLAVDIR) is the name of the PSLAVE working directory as seen by PEST. This directory name should terminate with a backslash character; if you do not terminate the name with a backslash, PEST will add it automatically. You can either provide the full path to the PSLAVE working directory, or supply it in abbreviated form if this works. Thus if, having opened a command-line window to run PEST, you transfer to drive K in Example 9.2 (this being a slave machine's disk) and change the directory to the PSLAVE working directory, and then change back to the local directory again (eg. by typing "`C:`"), then a path designation of `K:.\` affixed to the filenames listed in Table 9.1 will ensure that these files are written correctly to the PSLAVE working directory no matter what

is its full pathname. Note however that the directory `K:\` (without the dot) is not the same directory as `K:.\` (with the dot) if directories 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 Example 9.2 that a slave on the local machine can work from the same directory as PEST. This may be desirable if all model input files are in this same directory (which is often the case) and this is also the current PEST working directory. A designation of `.\` is sufficient to identify this directory.

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 parameter estimation 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 the 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 on all slave machines 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 directory 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 PEST control file read by Parallel PEST.)

Example 9.3 shows a Parallel PEST run management file equivalent to that of Example 9.2 but with the value of NFLETYP set to 0. Use of an abbreviated run management file such as that shown in Example 9.3 is only possible where all model input and output files on each slave reside in the one subdirectory, and this subdirectory is also the PSLAVE working directory on that machine.

```
prf
3 0 1.5
'my machine' .\
'steve''s machine' k:.\
'jerome''s machine' l:\model
600 600 720
```

Example 9.3 A Parallel PEST run management file equivalent to that of Example 9.2, but with NFLETYP set to zero.

9.2.6 More on Partial Parallelisation of the Marquardt Lambda Testing Process

The algorithm used by PEST to undertake parallel model runs as part of its lambda search procedure is similar 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 "packets" of 1 or more runs, and will not resume its normal execution until all of those runs are completed. The size of this "packet of runs" at any stage of the lambda search 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.

Another difference between the procedure by which Jacobian runs are carried out in parallel and that by which lambda search runs are carried out in parallel is that in the former case PEST knows the number of runs that must be carried out before the Jacobian calculation process is complete. However the lambda search procedure is deemed to be complete when PEST judges that the overall parameter estimation process is better served by terminating the current lambda search procedure and moving on to the next optimisation iteration. The criteria by which this decision is made are supplied through the variables appearing on the sixth line of the PEST control file. Hence the size of the "packet" of parallel model runs ordered by PEST is determined solely 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 search procedure. The decision-making process involved in the lambda search 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 optimisation iteration, upon commencement of the lambda search procedure for that optimisation 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 optimisation iteration. However, if there are enough slaves at its disposal, PEST also carries out model runs based on one or a number of higher Marquardt lambda values as well. On subsequent occasions during the same lambda search procedure on which PEST orders packets of model runs to be completed, parameters used for such runs are all based on decreasing Marquardt lambdas or on increasing Marquardt lambdas, depending on the results of the previous package of parallel runs.

The lambda search 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 (ie. 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 its lambda search algorithm. If the demands of that algorithm are such that more Marquardt lambdas must then be tested, another "packet" of runs is carried out.

Nevertheless, there will be some occasions when the path taken by the parameter estimation process is slightly different when the lambda search procedure is parallelised from that which is taken when the lambda search is conducted on the basis of serial model runs. If an unexpected and significant advance in the parameter estimation 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, PEST will not ignore this; the results of that run will be assimilated into the parameter estimation process.

9.3. Using Parallel PEST

9.3.1 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 IFLETYP 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 in the manner discussed in Section 9.2.5.

Before Parallel PEST is started, care should be taken to ensure that the model runs correctly on each slave machine. A set of model input files should be transferred from the master machine to each slave machine (or TEMPCHEK can be used to construct such files on the basis of a set of template files and a parameter value file). Where any model input files are not generated by Parallel PEST (because they contain no adjustable parameters), these files should be identical across all machines that run the model. In most cases all model input files will be placed into a single model working directory on each slave machine. In most cases the model itself will need to be installed on each slave machine as well.

9.3.2 Starting the Slaves

Go to each of the slaves in turn, open a command-line window and transfer to the PSLAVE working directory on that machine. Start PSLAVE execution by typing the command "pslave". PSLAVE immediately prompts the user for the command which it must use to run the model. Type in the appropriate command. Remember that, as with the 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 local machine, or does not reside in the PSLAVE working directory.

On each slave machine, 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.

9.3.3 Starting PEST

The next step is to start PEST. Move to the machine on which PEST resides, open a command-line window, transfer to 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.

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, the PSLAVE working directory on each slave machine in order to verify that it has access to each such directory. It also informs each PSLAVE that it has commenced execution and waits for a response from each of them. Once it has received all necessary responses it commences the parameter optimisation process.

Operation of Parallel PEST is very similar to that of PEST. However 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 search procedure by setting the PARLAM variable to zero), 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 optimisation process is complete or the user interrupts it by typing the PPAUSE, PSTOP or PSTOPST command while situated in the PPEST working directory within another command-line window; see Section 5.4.1 for further details. In the former case PEST execution can be resumed if the PUNPAUSE command is typed. Meanwhile the run record file can be examined by opening it with any text editor.

9.3.4 Re-Starting Parallel PEST

If the RSTFLE variable on the PEST control file is set to *restart*, a terminated Parallel PEST run may be restarted at any time from the beginning of the optimisation iteration during which it was interrupted. This can be achieved through entering the command:

```
ppest case /r
```

where *case.pst* is the PEST control file for the current case. Alternatively, execution can be re-commenced at that point at which calculation of the Jacobian matrix had most recently been completed by typing the command:

```
ppest case /j
```

PSLAVE must be started in the usual fashion on each slave machine before issuing either of these commands. 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 commencement of a new PEST run.

If PEST is restarted without the "/r" or "/j" switch, it will commence the optimisation process from the very beginning. Once again, if PSLAVE is already running on each of the slave machines (having been initially started for the sake of a previous Parallel PEST run), it need not be restarted on any of these machines. 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 PEST is restarted with the "/r" or "/j" switch, neither the PEST control file, nor any template or instruction file should have been altered from that supplied to 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 recommence a lengthy execution using more, less, and/or different slaves than 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 recommenced on each of the slave machines identified in that file before PEST execution is recommenced.

9.3.5 Parallel PEST Errors

As in normal PEST operation, Parallel PEST run-time error messages are written to both the screen and the run record file.

9.3.6 Losing Slaves

If, during the course of a Parallel PEST run, a slave machine drops out of the network, under many circumstances 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 one less slave at its disposal. However if the slave machine drops out just while PEST is trying to read a model output file, PEST will terminate execution with an error message.

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 PEST, PEST will be able to continue execution

using just that single slave, as long as the time of network failure did not coincide with the time at which PEST was reading a model output file from a slave machine.

9.3.7 The Parallel PEST Run Management Record File

Section 5.2 discusses in detail 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 provides this file with an extension of ".rmr" (for "Run Management Record"). Example 9.4 shows part of a Parallel PEST run management record file.

```
                  PEST RUN MANAGEMENT RECORD FILE: CASE VES2

SLAVE DETAILS:-

Slave Name                  PSLAVE Working Directory
----------                  ------------------------
"slave 1"                   k:.\model\
"slave 2"                   l:.\model\
"slave 3"                   m:.\model\


Attempting to communicate with slaves ....

- slave "slave 2" has been detected.
- slave "slave 3" has been detected.
- slave "slave 1" has been detected.


SLAVE MODEL INPUT AND OUTPUT FILES:-

Slave "slave 1" ----->

    Model input files on slave "slave 1":-
        k:.\model\ves.in1
        k:.\model\ves.in2

    Model output files on slave "slave 1":-
        k:.\model\ves.ot1
        k:.\model\ves.ot2
        k:.\model\ves.ot3

    Model command line for slave "slave 1":-
        ves


Slave "slave 2" ----->

    Model input files on slave "slave 2":-
        l:.\model\ves.in1
        l:.\model\ves.in2

    Model output files on slave "slave 2":-
        l:.\model\ves.ot1
        l:.\model\ves.ot2
        l:.\model\ves.ot3

    Model command line for slave "slave 2":-
        ves


Slave "slave 3" ----->

    Model input files on slave "slave 3":-
        m:.\model\ves.in1
        m:.\model\ves.in2

    Model output files on slave "slave 3":-
        m:.\model\ves.ot1
```

```
        m:.\model\ves.ot2
        m:.\model\ves.ot3

     Model command line for slave "slave 3":-
        ves


AVERAGE WAIT INTERVAL: 50 hundredths of a second.


                            RUN MANAGEMENT RECORD

RUNNING MODEL FOR FIRST TIME  ----->
   21:50:00.19:- slave "slave 1" commencing model run.
   21:55:05.00:- slave "slave 1" finished execution; reading results.


 OPTIMISATION ITERATION NO.  1 ----->

 Calculating Jacobian matrix: running model  5 times .....
   21:55:23.65:- slave "slave 1" commencing model run.
   21:55:33.92:- slave "slave 3" commencing model run.
   21:55:44.20:- slave "slave 2" commencing model run.
   22:00:05.79:- slave "slave 2" finished execution; reading results.
   22:00:17.77:- slave "slave 3" finished execution; reading results.
   22:00:29.47:- slave "slave 2" commencing model run.
   22:00:39.58:- slave "slave 3" commencing model run.
   22:00:50.07:- slave "slave 1" finished execution; reading results.
   22:05:11.83:- slave "slave 2" finished execution; reading results.
   22:05:43.70:- slave "slave 3" finished execution; reading results.

 Testing parameter upgrades .....
   22:06:01.69:- slave "slave 2" commencing model run.
   22:11:16.45:- slave "slave 2" finished execution; reading results.
   22:11:27.49:- slave "slave 2" commencing model run.
   22:16:19.63:- slave "slave 2" finished execution; reading results.
   22:16:35.95:- slave "slave 2" commencing model run.
   22:21:23.48:- slave "slave 2" finished execution; reading results.


 OPTIMISATION ITERATION NO.  2 ----->

 Calculating Jacobian matrix: running model  5 times .....
   22:21:45.07:- slave "slave 2" commencing model run.
   22:21:55.40:- slave "slave 3" commencing model run.
   22:22:25.05:- slave "slave 1" commencing model run.
   22:27:05.83:- slave "slave 2" finished execution; reading results.
   22:27:28.20:- slave "slave 3" finished execution; reading results.
   22:27:39.52:- slave "slave 2" commencing model run.
   22:27:52.63:- slave "slave 3" commencing model run.
   22:28:05.18:- slave "slave 1" finished execution; reading results.
   22:33:06.55:- slave "slave 2" finished execution; reading results.
   22:33:33.03:- slave "slave 3" finished execution; reading results.

 Testing parameter upgrades .....
   22:34:11.46:- slave "slave 2" commencing model run.
   22:39:36.82:- slave "slave 2" finished execution; reading results.
   22:39:48.09:- slave "slave 2" commencing model run.
   22:45:40.28:- slave "slave 2" finished execution; reading results.
   22:45:51.38:- slave "slave 2" commencing model run.
   22:50:43.30:- slave "slave 2" finished execution; reading results.


 OPTIMISATION ITERATION NO.  3 ----->

 Calculating Jacobian matrix: running model 10 times .....
   22:50:54.46:- slave "slave 2" commencing model run.
   22:51:14.68:- slave "slave 3" commencing model run.
                              .
                              .
```

Example. 9.4 Part of a Parallel PEST run management record file.

As is explained elsewhere in this manual, if PEST (or Parallel PEST) execution is re-commenced through use of the "/r" or "/j" command line switches, 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 recommences

execution as those which it employed in its previous life.

9.3.8 Running PSLAVE on the Same Machine as Parallel PEST

On many occasions of Parallel PEST execution, at least one of the slaves will be run on the same machine as that on which PEST is run. In most cases of PEST usage, PEST's tasks are small compared with those of the model; hence there is adequate capacity on one single-processor machine to run both a model and PEST without serious diminution of the performance of either. However it is wise to ensure that the command-line window running PSLAVE (and hence the model) is the active window (unless, of course, the user is running other, interactive, applications at the same time). In this way the window undertaking most of the work (ie. the window running the model) will receive the highest priority in the allocation of system resources.

9.3.9 Running Parallel PEST on a Multi-Processor Machine

Parallel PEST can be used to harness the full potential of a multi-processor machine. Such a machine can either be used on its own or as part of a network of other machines, some of which may possess multiple processors and some of which may not.

On a single dual processor machine, operation of Parallel PEST is identical to that described above for machines across a network. Three command-line windows must be opened, two of which are used to run PSLAVE and one of which is for the use of PEST. A different set of model input files, residing in a different subdirectory, must be prepared for the use of each instance of PSLAVE. Each PSLAVE must also have its own separate working directory which, for convenience, may just as well be the subdirectory holding its set of model input files. Note that each PSLAVE can invoke the same model executable.

9.3.10 The Importance of the WAIT Variable

The role of the WAIT variable was briefly discussed in Section 9.2.5. 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 PEST or PSLAVE screen sent by the operating system. In some cases, as mentioned in Section 9.2.5, this message demands an answer which, if not provided, can temporarily remove a particular slave from the Parallel PEST optimisation 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 9.1 after they have acted on 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 proceed 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 optimisation 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 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 will be 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.

9.3.11 If PEST will not Respond

As is mentioned above, if the PPAUSE, PSTOP or PSTOPST command is issued while the command-line version of Parallel PEST is running, PEST execution will be interrupted in the usual way. However unlike the single window version of 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 variable WAIT. 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 PEST.

There is, however, one particular situation that can result in a large elapsed time between the issuing of either of the above commands and the reception of a response from 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 a user types any of these commands while PEST is thus engaged, he/she may have to wait some time for PEST to respond. Meanwhile PEST records on the run management record that there is a :-

```
problem (either model or communications) in reading results from slave "xxx"
```

9.3.12 The Model

The model can be any executable or batch program which can be run from the command line.

9.4 An Example

Once PEST has been installed, a subdirectory of the main PEST directory called *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 directory is cited in the PATH environment variable.

Change the working directory to the *ppestex* subdirectory and create two subdirectories to this directory called *test1* and *test2*.

Now open three command-line windows (you probably have one open already). In one of these windows transfer to subdirectory *test1* and type the command "pslave". When prompted for the command to run the model, type "..\a_model". Do the same in another command-line window for subdirectory *test2*.

In the third command-line window transfer to the directory holding the example files (ie. the parent directory of *test1* and *test2*) and type the command "ppest test". Parallel PEST should commence execution and, after verifying that it can communicate with each of its slaves, undertake parameter optimisation for the *a_model* model.

If you wish, you can also carry out the parameter estimation process using the single window version of PEST. While situated in the *ppestex* subdirectory, type "pest test". For this particular case the single window version of PEST will run faster than Parallel PEST. This is because the model run time is too small to justify Parallel PEST's run management overheads. Furthermore, unless you are using a multi-processor machine, there is nothing to be gained by undertaking parallel model runs on a single machine anyway. Note, however, that Parallel PEST's speed can be increased somewhat by reducing the value of the WAIT variable from that provided on the *test.rmf* file.

9.5 Frequently Asked Questions

For more information on Parallel PEST see the frequently asked questions listed in Chapter 13.

10. PEST Utilities

PEST is accompanied by five utility programs whose role is to assist in PEST input file preparation. These are programs TEMPCHEK, INSCHEK, PESTCHEK, PESTGEN and PARREP. The first three of these programs are used to check template files, instruction files and the PEST control file respectively for a particular PEST case, prior to actually running PEST on that case. In this way you can be sure that the input dataset which you supply to PEST is correct and consistent. The PESTGEN utility creates a PEST control file using default values for many of the PEST input variables. PARREP builds a new PEST control file based on an existing PEST control file and a set of values cited in a parameter value file.

A sixth utility program, named JACWRIT, acts in a PEST postprocessing capacity. It allows a user to generate and ASCII file in which the Jacobian matrix for a particular parameter estimation problem is recorded for inspection.

The PAR2PAR utility is used to undertake mathematical operations of arbitrary complexity between existing parameters in order to generate new parameters. It is normally used in a model preprocessing capacity, being run by PEST as part of a composite model encapsulated in a batch file. Its many uses include complex parameter transformation (perhaps to improve model linearity), and the generation of a large number of "secondary parameters" (as used by the model) from a smaller number of "primary parameters" (as estimated by PEST).

10.1 TEMPCHEK

Program TEMPCHEK checks that PEST template files obey PEST protocol. If provided with a set of parameter values, TEMPCHEK can also be used to generate a model input file from a template file. It builds the model input file in the same way that PEST does; you can then run your model, checking that it is able to read such a PEST-generated input file without any difficulties.

TEMPCHEK is run using the command

```
tempchek tempfile [modfile [parfile]]
```

where

tempfile is the name of a template file,
modfile is the name of a model input file to be generated by TEMPCHEK (optional), and
parfile is the name of a PEST parameter value file (also optional).

The simplest way to run TEMPCHEK is to use the command

```
tempchek tempfile
```

When invoked in this way TEMPCHEK simply reads the template file *tempfile*, checking it for breaches of PEST protocol. It writes any errors it finds to the screen. These errors can be redirected to a file using the ">" symbol on the TEMPCHEK command line. Thus to run program TEMPCHEK, directing it to write any errors found in the template file *model.tpl* to the file *errors.chk*, use the following command

```
tempchek model.tpl > errors.chk
```

If no errors are encountered in the template file, TEMPCHEK informs you of this through an appropriate screen message. This message also informs you of the number of parameters that TEMPCHEK identified in the template file. TEMPCHEK lists these parameters in a file named *file.pmt*, where *file* is the filename base of *tempfile* (ie. the filename minus its extension). If *tempfile* has no extension TEMPCHEK simply adds the extension ".pmt" to *tempfile*. By supplying a parameter value as well as a scale and offset for each parameter, *file.tmp* can be transformed into a PEST parameter value file which TEMPCHEK can then use to generate a model input file (see below).

Note that if a parameter is cited more than once in a template file, the parameter is nevertheless written only once to *file.pmt*; also it is counted only once as TEMPCHEK sums the total number of parameters cited in the template file.

If you wish TEMPCHEK to generate a model input file you must supply it with the name of the template file upon which the model input file is based, the name of the model input file which it must generate, and the values of all parameters named in the template file. To run TEMPCHEK in this fashion, enter the command

```
tempchek tempfile modfile [parfile]
```

The name of the parameter value file is optional. If you don't supply a name TEMPCHEK generates the name itself by replacing the extension used in the template filename with the extension ".par"; if *tempfile* has no extension, ".par" is simply appended. Hence the naming convention of the parameter value file is in accordance with that used by PEST which generates such a file at the end of every optimisation iteration; see Section 5.3.1.

A PEST parameter value file is shown in Example 5.2. The first line of a parameter value file must contain values for the character variables PRECIS and DPOINT; the role of these variables is discussed in Section 4.2.2. These variables must be supplied to TEMPCHEK so that it knows what protocol to use when writing parameter values to the model input file which it generates.

The second and subsequent lines of a parameter value file each contain a parameter name, a value for the named parameter, and the scale and offset to be used when writing the parameter value to the model input file. Because TEMPCHEK is supplied with a scale and offset for each parameter, it is able to generate model input files in exactly the same way that PEST does; see Section 4.2.4.

If TEMPCHEK finds a parameter in a template file which is not listed in the parameter value file, it terminates execution with an appropriate error message. However a parameter value file may contain more parameters than are cited in the template file; these extra parameters are ignored when generating the model input file. This may occur if your model has a number of input files and you wish to optimise parameters occurring on more than one of them. You must make a template file for each such model input file; however you need to prepare only one parameter value file containing all the parameters for that particular problem.

10.2 INSCHEK

Program INSCHEK assists in the construction of PEST instruction files. Like TEMPCHEK it can be used in two modes. In the first mode it simply checks that an instruction file has no syntax errors and obeys PEST protocol as set out in Section 3.3. In its second mode it is able to read a model output file using the directions contained in the instruction file; it then writes a file listing all observations cited in the instruction file together with the values of these observations as read from the model output file. In this way you can verify that not only is your instruction set syntactically correct, but that it reads a model output file in the way it should.

INSCHEK is run using the command

```
inschek insfile [modfile]
```

where

insfile is a PEST instruction file, and
modfile is a model output file to be read by INSCHEK (optional).

The simplest way to run INSCHEK is to use the command

```
inschek insfile
```

When invoked in this way, INSCHEK simply reads the instruction file *insfile*, checking that every instruction is valid and that the instruction set is consistent. If it finds any errors it writes appropriate error messages to the screen. You can redirect this screen output to a file if you wish by using the ">" symbol on the command line. Thus to run INSCHEK such that it records any errors found in the instruction file *model.ins* to the file *errors.chk*, use the command

```
inschek model.ins > errors.chk
```

If no errors are found in the instruction file *insfile*, INSCHEK informs you of how many observations it identified in the instruction set and lists these observations to *file*.*obf*, where *file* is the filename base (ie. the filename without its extension) of *insfile*; if *insfile* has no extension, the extension ".obf" is simply appended to the filename.

For an instruction set to be useable by PEST it must do more than simply obey PEST protocol; it must also read a model output file correctly. You can check this by invoking INSCHEK with the command

```
inschek insfile modfile
```

When run in this way, INSCHEK first checks *insfile* for syntax errors; if any are found it writes appropriate error messages to the screen and does not proceed to the next step. Alternatively, if the instruction set contained in *insfile* is error free, INSCHEK reads the model output file *modfile* using the instruction set. If any errors are encountered in this process, INSCHEK generates an appropriate error message and abandons execution; such errors may arise if, for example, INSCHEK finds a blank space where a number should be, encounters the end of the model output file before locating all observations, etc. However if

INSCHEK reads the file without trouble, it lists all observations cited in the instruction set, together with their values as read from *modfile*, to *file.obf*, where *file* is the filename base of *insfile*. Example 10.1 shows a typical observation value file.

```
ar1       1.21038
ar2       1.51208
ar3       2.07204
ar4       2.94056
ar5       4.15787
ar6       5.77620
ar7       7.78940
ar8       9.99743
ar9       11.8307
ar10      12.3194
ar11      10.6003
ar12      7.00419
ar13      3.44391
ar14      1.58278
ar15      1.10381
ar16      1.03085
ar17      1.01318
ar18      1.00593
ar19      1.00272
```

Example 10.1 An observation value file.

10.3 PESTCHEK

PESTCHEK should be used when all preparations for a PEST run are complete, ie. when all template files, instruction files and the PEST control file which "brings it all together" have been prepared. PESTCHEK reads the PEST control file, making sure that all necessary items of information are present on this file and that every item is consistent with every other item (for example that logarithmically-transformed parameters do not have negative lower bounds, that RELPARMAX is greater than unity if at least one parameter is free to change sign during the optimisation process etc.). As PEST does not carry out consistency checks such as these, it is essential that PESTCHEK be used to check all input data prior to a PEST run.

PESTCHEK also carries out some of the tasks undertaken by programs TEMPCHEK and INSCHEK, viz. it checks all template and instruction files cited in the PEST control file for correct syntax. Unlike TEMPCHEK and INSCHEK, PESTCHEK cannot generate a model input file nor read a model output file; nevertheless it does check that all parameters and observations cited in the PEST control file are also cited in the template and instruction files referenced in the PEST control file, and that parameters and observations cited in template and instruction files are also listed in the PEST control file.

PESTCHEK is run using the command

```
pestchek case
```

where

case is the filename base of a PEST control file.

No filename extension should be provided here; an extension of ".pst" is added automatically. This is the same filename base which should be provided to PEST on its command line; see Section 5.1.2. PESTCHEK reads an identical dataset to PEST.

PESTCHEK writes any errors it encounters to the screen. If you wish, error messages can be redirected to a file using the ">" symbol on the PEST command line. Thus to check the dataset contained in the PEST control file, *calib.pst*, and the template and instruction files cited therein, directing any error messages to the file *errors.chk*, invoke PESTCHEK using the command

```
pestchek calib > errors.chk
```

If PESTCHEK finds one or a number of errors in your input dataset it is important that you re-run PESTCHEK on the dataset after you have corrected the errors. This is because PESTCHEK may not have read all of your input dataset on its first pass; depending on the errors it finds, it may not be worthwhile (or possible) for PESTCHEK to read an input dataset in its entirety once an error condition has been established. Hence, once you have rectified any problems that PESTCHEK may have identified in your input dataset, you should submit it to PESTCHEK again, being content that the data is fully correct and consistent only when PESTCHEK explicitly informs you that this is the case.

If you wish, you can write a batch file which runs both PESTCHEK and PEST in sequence. Because PESTCHEK terminates execution with a non-zero errorlevel setting should it detect any errors, you can program the batch process to bypass the running of PEST unless the input dataset is perfect. In this way, you can always be sure that PESTCHEK, rather than PEST, is the first to detect any input data errors. A suitable batch file is shown in Example 10.2.

```
@ echo off
rem FILE RUNPEST.BAT
rem To run RUNPEST.BAT type the command "runpest case [/r] [/j]",
rem where case is the filename base of the PEST control file, and
rem "/r" and "/j" are optional restart switches.
pestchek %1
if errorlevel 1 goto end
pest %1 %2
:end
```

Example 10.2 Running PESTCHEK and PEST as a batch process.

10.4 PESTGEN

Program PESTGEN generates a PEST control file. In most cases this file will need to be modified before PEST is run, as PESTGEN generates default values for many of the PEST input variables supplied on this file; it is probable that not all of these default values will be appropriate for your particular problem.

PESTGEN is run using the command

```
pestgen case parfile obsfile
```

where

case is the case name. No filename extension should be supplied; PESTGEN automatically adds the extension ".pst" to *case* in order to form the filename of the PEST control file which it writes.

parfile is a parameter value file, and

obsfile is an observation value file.

A parameter value file is shown in Example 5.2; Example 10.1 shows an observation value file. The former file must include all parameters used in the current case; these parameters may be cited in one or a number of template files. Similarly, the observation value file must provide the name and value for all observations used in the current problem; the observations, too, may be cited on one or a number of instruction files. The observation values provided in this file may be field/laboratory measurements or, if PEST is being run on theoretical data, model-generated observation values. In the latter case program INSCHEK may be used to generate the file; if there are multiple model output files, observation value files generated on successive INSCHEK runs could be concatenated to form an appropriate observation value file to provide to PESTGEN.

PESTGEN commences execution by reading the information contained in files *parfile* and *obsfile* (see above), checking them for correctness and consistency. If there are any errors in either of these files, PESTGEN lists these errors to the screen and terminates execution. Alternatively, if these files are error-free, PESTGEN then generates a PEST control file.

Files *parfile* and *obsfile* provide PESTGEN with the names of all parameters and observations which need to be listed in the PEST control file. They also provides PEST with initial parameter values (these must be provided in the second column of the parameter value file), the scale and offset for each parameter (in the third and fourth columns of the parameter value file), the laboratory or field measurement set (in the second column of the observation value file) and values for the variables PRECIS and DPOINT (on the first line of the parameter value file). For all other variables listed in the PEST control file, PESTGEN uses default values.

For the parameter and observation value files shown in Examples 5.2 and 10.1, the PESTGEN-generated PEST control file is shown in Example 10.3.

```
pcf
* control data
restart estimation
    5    19     5     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
ro1  relative 0.01  0.0  switch  2.0 parabolic
ro2  relative 0.01  0.0  switch  2.0 parabolic
ro3  relative 0.01  0.0  switch  2.0 parabolic
h1   relative 0.01  0.0  switch  2.0 parabolic
h2   relative 0.01  0.0  switch  2.0 parabolic
* parameter data
ro1  none relative    1.00000      -1.00000E+10   1.00000E+10 ro1     1.0000     0.00000 1
ro2  none relative    40.0009      -1.00000E+10   1.00000E+10 ro2     1.0000     0.00000 1
ro3  none relative    1.00000      -1.00000E+10   1.00000E+10 ro3     1.0000     0.00000 1
h1   none relative    1.00000      -1.00000E+10   1.00000E+10 h1      1.0000     0.00000 1
h2   none relative    9.99978      -1.00000E+10   1.00000E+10 h2      1.0000     0.00000 1
* observation groups
obsgroup
* observation data
ar1    1.21038      1.0  obsgroup
ar2    1.51208      1.0  obsgroup
ar3    2.07204      1.0  obsgroup
ar4    2.94056      1.0  obsgroup
ar5    4.15787      1.0  obsgroup
ar6    5.77620      1.0  obsgroup
ar7    7.78940      1.0  obsgroup
ar8    9.99743      1.0  obsgroup
ar9    11.8307      1.0  obsgroup
ar10   12.3194      1.0  obsgroup
ar11   10.6003      1.0  obsgroup
ar12   7.00419      1.0  obsgroup
ar13   3.44391      1.0  obsgroup
ar14   1.58278      1.0  obsgroup
ar15   1.10381      1.0  obsgroup
ar16   1.03085      1.0  obsgroup
ar17   1.01318      1.0  obsgroup
ar18   1.00593      1.0  obsgroup
ar19   1.00272      1.0  obsgroup
* model command line
model
* model input/output
model.tpl  model.inp
model.ins  model.out
* prior information
```

Example 10.3 A PEST control file generated by PESTGEN.

Note that when viewing a PESTGEN-generated PEST control file on your screen, the OFFSET values in the "parameter data" section of the file may not be visible as they are written beyond the 80th column of the file; to bring them into view, move your editor's cursor over them.

Example 10.3 shows the default values used by PESTGEN in generating a PEST control file. The following features, in particular, should be noted.

- PESTGEN assumes that PEST will be run in parameter estimation mode. Neither a "predictive analysis" nor a "regularisation" section is included in the PEST control

file.

- PESTGEN generates a separate parameter group for each parameter; the name of the group is the same as that of the parameter. For each of these groups derivatives are calculated using a relative increment of 0.01, with no absolute lower limit provided for this increment. At the beginning of the optimisation process, derivatives will be calculated using the forward method, switching to the three-point "parabolic" method on the iteration following that for which the objective function fails to undergo a relative reduction of at least 0.1 (ie. PHIREDSWH). The derivative increment to be used in implementing the "parabolic" method is twice the increment used in implementing the forward method of derivatives calculation.

- No prior information is supplied.

- No parameters are tied or fixed; no parameters are log-transformed and changes to all parameters are relative-limited (with a RELPARMAX value of 3.0). The upper bound for each parameter is provided as 1.0E10, while the lower bound is -1.0E10. It is strongly suggested that you modify these bounds to suit each parameter. It is also recommended that you consider log-transforming some parameters for greater optimisation efficiency; see Section 2.2.1. Note, however, that the lower bound of a log-transformed parameter must be positive and that its changes must be factor-limited.

- All observations are provided with a weight of 1.0.

- PESTGEN assumes that the model is run using the command "model". It also assumes that the model requires one input file, viz. *model.inp*, for which a template file *model.tpl* is provided. It further assumes that all model-generated observations can be read from one output file, viz. *model.out*, using the instructions provided in the instruction file *model.ins*. You will almost certainly need to alter these names. If there are, in fact, multiple model input and/or output files, don't forget to alter the variables NTPLFLE and NINSFLE in the "control data" section of the PEST control file.

- The default values for all other variables can be read from Example 10.3.

Once you have made all the changes necessary to the PESTGEN-generated PEST control file, you should check that your input dataset is complete and consistent using program PESTCHEK. If PESTCHEK informs you that all is correct, then you are ready to run PEST.

10.5 PARREP

Program PARREP replaces initial parameter values as provided in a PEST control file by another set of values, the latter being supplied in a PEST parameter value file.

Recall from Section 5.3.1 that in the course of the parameter estimation process PEST writes a parameter value file every time it improves its parameter estimates. After a PEST run has finished (either of its own accord or if it was manually halted), optimised parameter values can be found in the parameter value file. The parameter value file possesses the same filename base as the current case but has an extension of ".par". Because it has such a simple

structure, a parameter value file can also be easily built by the user with the help of a text editor.

PARREP is useful when commencing a new PEST run where an old run finished. An updated PEST control file can be produced by replacing parameter values in the old file with the best parameter values determined during the previous PEST run as recorded in the parameter value file written during that run. Recommencing a PEST run in this way, rather than through use of the "/r" or "/j" switches, allows a user to alter certain PEST control variables, fix or tie certain parameters, or adjust PEST's management of the parameter estimation process in other ways, prior to re-commencement of the run.

PARREP is also useful when undertaking a single model run on the basis of a certain set of parameters in order to calculate the objective function. Simply modify an existing PEST control file using PARREP as described above, and set NOPTMAX to zero.

PARREP is run using the command:

```
parrep parfile pestfile1 pestfile2
```

where

parfile is the name of a parameter value file,
pestfile1 is the name of an existing PEST control file, and
pestfile2 is the name for the new PEST control file.

When PARREP replaces parameter values in the existing PEST control file by those read from the parameter value file, it does not check that each parameter value lies between its upper and lower bounds, that log-transformed parameters are positive, etc. Hence, especially if using a manually-created parameter value file, it is, as always, a good idea to run PESTCHEK before running PEST to ensure that all is consistent and correct.

10.6 JACWRIT

JACWRIT is a utility program which allows the user to inspect the Jacobian matrix computed by PEST. Recall from Chapter 2 that the Jacobian matrix contains the derivative of each model output for which there is a corresponding observation with respect to each parameter.

At the end of each optimisation iteration PEST records a binary file containing the Jacobian matrix corresponding to "best" parameters so far attained during the optimisation process. The definition of "best" depends on the aim of the optimisation process. When working in parameter estimation mode the best parameters are those for which the lowest objective function was obtained. When working in predictive analysis mode, they are those for which the prediction was maximised/minimised compatible with the objective function being below the user-supplied calibration threshold. When working in regularisation mode, the best parameters are those for which the regularisation objective function is the least, provided that the measurement objective function is below the user-supplied measurement objective function limit. The name of the binary file in which the Jacobian matrix is stored is *case.jco* where *case* is the filename base of the current PEST control file; "jco" stands for "Jacobian optimised".

The Jacobian file is stored in binary rather than text format to save space. To translate it to text format, you must run JACWRIT by typing the command:-

```
jacwrit jacfile1 jacfile2
```

where

jacfile1 is the name of the binary Jacobian file written by PEST, and
jacfile2 is the name of the text file to which JACWRIT should write the Jacobian matrix in a form which is fit for human consumption.

Note the following:-

- Parameter and observation names are listed in the text file written by JACWRIT so that it becomes an easy matter to link a sensitivity (ie. a derivative) to a particular parameter/observation pair.
- Only adjustable parameters are represented in the file written by JACWRIT; fixed and tied parameters are not represented.
- The sensitivity of a parameter to which another parameter is tied reflects the fact that this parameter "carries" at least one other parameter through the optimisation process.
- Derivatives reflect the transformation status of a parameter. Thus if a parameter is log-transformed, the derivative with respect to the log of that parameter is presented.

10.7 PAR2PAR

10.7.1 General

On many occasions of model calibration there is a need to manipulate parameters before providing them to a model. There can be a number of reasons for this; two of them are now outlined.

10.7.1.1 Parameter Ordering

Suppose that a particular model has three parameters named *infilt1*, *infilt2* and *infilt3*. For purposes of illustration, let it be assumed that these parameters govern infiltration of water into different parts of a catchment, in this case into subareas 1, 2 and 3 respectively. Soil property data may suggest that infiltration increases with subarea index, that is that *infilt1* < *infilt2* < *infilt3*. Thus, during the parameter estimation process, it would be desirable for the lower bound of *infilt2* to be the current value for *infilt1*, and for the lower bound of *infilt3* to be the current value of *infilt2*.

Unfortunately it would be very difficult to incorporate parameter-dependent bounds into the PEST inversion algorithm. However an alternative path can be taken which accomplishes the same thing. This alternative path consists of estimating *infilt1* together with two other parameters named *infiltrat2* and *infiltrat3* ("*infiltrat*" stands for "infiltration ratio"). These latter two parameters are defined by the relationships:-

$$infiltrat2 = infilt2/infilt1 \qquad (10.1a)$$

and

$$infiltrat3 = infilt3/infilt2 \qquad (10.1b)$$

Desired infiltration parameter ordering relationships will be maintained if each of *infiltrat2* and *infiltrat3* is provided with a lower bound of 1.0 in the parameter estimation process implemented by PEST.

In using this device to ensure that correct infiltration parameter ordering relationships are maintained, PEST must "see" parameters *infilt1*, *infiltrat2* and *infiltrat3*, while the model must "see" parameters *infilt1*, *infilt2* and *infilt3*. The necessary "parameter transformation" process can be accomplished by running the utility program PAR2PAR as a model preprocessor contained in a "composite model" run by PEST as a batch file. PAR2PAR "receives" the current PEST-calculated values of *infilt1*, *infiltrat2* and *infiltrat3*; it then "transforms" these into values for *infilt1*, *infilt2* and *infilt3*. Then it writes one or more model input files (based on appropriate template files) containing the current values of these native model parameters. Based on equations 10.1a and 10.1b, PAR2PAR must be "programmed" to calculate *infilt2* and *infilt3* using the relationships:-

$$infilt2 = infilt1 * infiltrat2$$

$$infilt3 = infilt2 * infiltrat3$$

10.7.1.2 Seasonal Parameter Variations

Some model parameters show seasonal variation. For environmental models which simulate water or crop-growth processes in agricultural areas, "crop factor" may be one such parameter. Crop factor is also a parameter that (together with other parameters) often requires adjustment through the calibration process in order that the model can replicate measured crop water usage, observed crop growth, or some other system response for which historical records are available.

Many models require that the crop factor be provided on a monthly basis. However while monthly crop factors may indeed require estimation through the calibration process, it would generally be unwise to attempt to estimate each monthly crop factor independently of every other monthly crop factor through the calibration process, for this would ignore an inherent relationship between these parameters, this being the fact that variation of crop factor with season may show a regular (perhaps sinusoidal) pattern. To ignore this pattern in parameterising the model would be to ignore an important facet of system behaviour. Furthermore, in many model calibration contexts, it would be unlikely that 12 different monthly crop factors could be independently estimated with any degree of uniqueness because of the high degree of correlation that is likely to exist between these individual parameters (especially where the data available for model calibration is limited).

For a case such as this, a suitable parameterisation strategy may be to estimate the mean monthly crop factor, together with the amplitude and phase of the seasonal variation of the crop factor about this mean. Thus twelve parameters are replaced with three. This will lend stability to the parameter estimation process as it promulgates a more unique solution to it. In implementing this strategy, PEST will "see" three parameters while the model will still "see"

the twelve parameters which it requires. The task of "transforming" the three parameters "seen" by PEST to the twelve parameters "seen" by the model can be accomplished using PAR2PAR as a model preprocessor, run by PEST just before the model on every occasion that the model is run. Once again, this can be accomplished by including both of the PAR2PAR and model executables in a batch file run by PEST as a "composite model". On the basis of the three parameters adjusted by PEST (named, for example, *mean*, *amplitude* and *phase*), PAR2PAR will calculate the monthly crop factor parameters required by the model (named, for example, *crop1, crop2…crop12*) using a series of relationships such as:-

*crop1 = mean + amplitude * sin ((1 + phase)*2.0*3.142/12.0)*

*crop2 = mean + amplitude * sin((2 + phase)*2.0*3.142/12.0)*

etc

In these equations *phase* is measured in months; as is explained below, the argument of the *sin* function must be supplied in radians, where 2π radians is equal to a full cycle.

Seasonal parameter variation can be expressed in a number of different ways; use of the *sin* function is just one of them. Another method would be to use "seasonal ratios"; in this case only one parameter may require estimation, this being the factor by which all such ratios are multiplied to achieve model calibration.

10.7.2 Using PAR2PAR

10.7.2.1 Running PAR2PAR

PAR2PAR is run using the command:-

```
par2par infile
```

where *infile* is a PAR2PAR input file which must be prepared by the user.

10.7.2.2 The PAR2PAR Input File

The structure of the PAR2PAR input file is shown in Example 10.4. An example of such a file is provided in Example 10.5.

```
* parameter data
PARNME = expression
PARNME = expression
.
.
* template and model input files
TEMPFLE INFLE
TEMPFLE INFLE
.
.
* control data
PRECIS DPOINT
```

Example 10.4 Structure of the PAR2PAR input file.

```
* parameter data
infilt1 = 0.3456
infiltrat2 = 1.0453
infiltrat3 = 1.5432
infilt2= infilt1 * infiltrat2
infilt3 = infilt2 * infiltrat3
* template and model input files
model1.tpl model1.in
model2.tpl model2.in
* control data
single point
```

Example 10.5. An example of a PAR2PAR input file.

A PAR2PAR input file must contain at least a "parameter data" section and a "template and model input files" section. The "control data" section is optional; if it is omitted, the default values of "single" and "point" are supplied for the variables PRECIS and DPOINT.

The "parameter data" section of the PAR2PAR input file provides the means whereby values are assigned to a set of parameters. These values can be provided either by the direct assignment of numbers, or through mathematical expressions. These expressions (which may be of considerable complexity) may cite parameters whose values were assigned in previous expressions.

The "template and model input files" section of the PAR2PAR input file provides the names of template files together with the names of the model input files to which they correspond. Once it has determined values for all parameters appearing on the left sides of the expressions listed in the "parameter data" section of its input file, PAR2PAR writes these parameter values to the nominated model input files using template files based on these model input files (just like PEST does). Note the following:-

- Any parameter appearing in any of the template files listed in the "template and

model input files" section of the PAR2PAR input file must be assigned a value in the "parameter data" section of the PAR2PAR input file.

- If there is more than one template/model input file pair listed in the "template and model input files" section of the PAR2PAR input file, any particular template file can be cited more than once if desired. However each model input file can be cited only once, for it would make no sense for a model input file generated on the basis of one template file to be overwritten by another model input file generated on the basis of the same or another template file.

If either of these rules are violated, PAR2PAR will inform you of this through an appropriate error message.

All template files cited in the "template and model input files" section of the PAR2PAR input file should be checked for correctness using TEMPCHEK. While PAR2PAR will detect and report any errors that it finds in these files, it will only report the first error that it encounters; then it will cease execution. TEMPCHEK, on the other hand, attempts to examine the entirety of a template file, reporting all errors to the screen.

10.7.2.3 Parameter Relationships

The relationships by which parameter values are calculated from numbers, or from values previously assigned to other parameters, may be mathematical expressions of complex form. They can include any or all of the "*", "/", "+", "-" and "^" operators as well as brackets. (Note that the "^" operator raises the number in front of the "^" symbol to a power equal to the number trailing the "^" symbol; this operation can also be designated using the "**" symbol as in the FORTRAN programming language.) Mathematical operations of equal rank are evaluated in the order "^" followed by "*" and "/", followed by "+" and "-", as is the usual convention. This order can be overridden by the use of brackets.

The following mathematical functions are supported in expressions by which parameter values are calculated – *sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sinh*, *cosh*, *tanh*, *exp*, *log*, *log10*, *abs* and *sqrt*. Note the following rules governing use of these functions:-

- As is the FORTRAN convention, the arguments of the trigonometric functions *sin*, *cos* and *tan*, and the values returned by their inverse functions *asin*, *acos* and *atan*, are assumed to be in radians. There are 2π radians in a circle; thus 2π radians are equal to 360 degrees.

- The *log* function is to base *e*; for logarithms to base 10, use the *log10* function.

- For some of the functions listed above, arguments must lie within a specific numerical range (for example the argument of the *log* function must always be greater than zero). If a function argument is provided which is outside of its legal range, PAR2PAR will often trap the error and cease execution with an appropriate error message. However in some rare instances the argument may "slip through" and a compiler-generated error message will be supplied upon termination of PAR2PAR execution.

The following rules apply when formulating mathematical expressions to calculate parameter values.

- Expressions may contain both numbers and parameters. However where a parameter is used, its value must have been calculated (or supplied) in a previous expression.
- As is the normal PEST convention, parameter names must be 12 characters or less in length.
- Spaces can be placed next to operators, brackets and functions. However they cannot appear within numbers, parameter names or function names.

Some examples of allowable mathematical expressions follow:-

*trans5 = k5 * (top5 – bottom5)*

pi = 3.14159

*par3 = 3.4 * (4.5 + trans5 ^ (3 + sin(0.6)))*

par4 = par3 / (pi + exp(5.0 + par3/trans5))

*par5 = -(par1 + par2) * cosh(pi * trans5)*

If an expression is long, it may be continued onto the next line by placing the "&" character at the beginning of that line. Thus the expression:-

*par5 = -(par1 + par2) * cosh(pi * trans5)*

is equivalent to:-

par5 =

& -(par1+par2)

*& * cosh(*

*& pi * trans5)*

10.7.2.4 Generation of Model Input Files

Once it has calculated values for all parameters, PAR2PAR writes these values to one or more model input files using templates of these files to govern parameter value placement. Use of template files for writing model input files is fully discussed in Chapter 3 of this manual. As is described in that chapter, slight variations of the way in which numbers representing parameter values are written to model input files can be effected through use of the PRECIS and DPOINT variables; values for these variables are supplied in the optional "control data" section of a PAR2PAR input file. If PRECIS is set to "single", numbers are written to model input files using the "E" character for exponentiation. However if it is set to "double", the "D" character is used. Furthermore, if there is sufficient space, up to 23

characters can be used to record the value of the parameter instead of the usual maximum of 13. Setting the DPOINT variable to "nopoint" instructs PEST to write a parameter's value to a model input file without the decimal point if this can be accomplished through numerical formatting, thereby gaining one extra significant figure of precision (more will be said about precision shortly). As is stated above, the "control data" section of the PAR2PAR input file is optional; if it is omitted, default values of "single" and "point" are supplied for PRECIS and DPOINT respectively.

10.7.3 Using PAR2PAR with PEST

10.7.3.1 The Composite Model

As was discussed above, when used with PEST, PAR2PAR will normally be run as part of a "composite model" encapsulated in a batch file. Thus whenever PEST runs the model, it first runs PAR2PAR (and any other model preprocessors cited in the batch file), followed by the model (followed by any model postprocessors cited in the batch file).

As for any other model executable program which uses parameters which require estimation by PEST, a template file must be built, based on a PAR2PAR input file. Just before it runs the model, PEST will then write current parameter values to the PAR2PAR input file using the corresponding template file. An example of such a template file, based on the PAR2PAR input file shown in Example 10.5, is provided in Example 10.6.

```
ptf $
* parameter data
infilt1 = $infilt1  $
infiltrat2 = $infiltrat2$
infiltrat3 = $infiltrat3$
infilt2= infilt1 * infiltrat2
infilt3 = infilt2 * infiltrat3
* template and model input files
model.tpl model.in
* control data
single point
```

Example 10.6. A template for the PAR2PAR input file of Example 10.5.

Based on the template file of Example 10.6, before PEST runs the model it will replace the strings "$infilt1$", "$infiltrat2$", and "$infiltrat3$" with the current values of these parameters. Note that these parameters do not need to be named the same as the PAR2PAR parameters to which values are assigned in the pertinent expressions in the PAR2PAR input file. They could have been given any name at all; the same parameter names are used by both PEST and PAR2PAR in this example simply as a matter of convenience. Furthermore, parameter spaces in the template of a PAR2PAR input file do not need to be restricted in their location to the right side of expressions comprised of a "=" symbol followed by a single number. See, for example, the PAR2PAR input file and corresponding template file depicted in Examples 10.7 and 10.8. These accomplish the same task as the files depicted in Examples 10.5 and 10.6.

```
* parameter data
infilt1 = 0.3456
infilt2= infilt1 * 1.23983
infilt3 = infilt2 * 1.53953
* template and model input files
model.tpl model.in
```

Example 10.7. A PAR2PAR input file.

```
ptf $
* parameter data
infilt1 = 0.3456
infilt2= infilt1 * $infiltrat2$
infilt3 = infilt2 * $infiltrat3$
* template and model input files
model.tpl model.in
```

Example 10.8. A template for the PAR2PAR input file of Example 10.7.

It is apparent that when using PAR2PAR as part of a composite model run by PEST there are two sets of template files involved in the inversion process, viz. that used by PEST to write a PAR2PAR input file, and those used by PAR2PAR to write model input files. These should not be confused. PEST should never be instructed to use a template file to write a model input file that is also cited in the "template and model input files" section of a PAR2PAR input file. If this happens, the model input file generated by PEST will be overwritten by that generated by PAR2PAR.

It often happens that only a few parameters required by a model need to be calculated by an expression cited in a PAR2PAR input file; other model parameters can be estimated directly by PEST. These latter parameters can simply be "passed through" PAR2PAR by assigning them numerical values in the pertinent expressions in the PAR2PAR input file. Example 10.9 shows a PAR2PAR input file in which only parameter *par8* is calculated through manipulation of other parameters; Example 10.10 shows the corresponding template file of the PAR2PAR input file. Parameters *par1* to *par5* are passed directly to the model through the template file *model.tpl* of the model input file *model.in*. The template file *model.tpl* thus cites all of parameters *par1* to *par5* as well as parameter *par8*. (It may also cite *par6* and *par7*.)

```
* parameter data
par1 = 1.583745e-4
par2 = 5.395832e-1
par3 = 4.583924e-2
par4 = 5.389028e-5
par5 = 4.389428e-2
par6 = 3.559313e-1
par7 = 5.395355e-2
par8 = par6 * exp(-par7)
* template and model input files
model.tpl model.in
```

Example 10.9. A PAR2PAR input file.

```
ptf $
* parameter data
par1 = $par1      $
par2 = $par2      $
par3 = $par3      $
par4 = $par4      $
par5 = $par5      $
par6 = $par6      $
par7 = $par7      $
par8 = par6 * exp(-par7)
* template and model input files
model.tpl model.in
```

Example 10.10. A template file for the PAR2PAR input file of Example 10.9.

10.7.3.2 Numerical Precision

As is explained elsewhere in this manual, when PEST writes a number to a model input file on the basis of a template file, it alters its internal representation of that number to account for the fact that the number may be written to the model input file with less than the maximum number of significant figures with which that number can be represented internally within the computer. Thus when PEST calculates derivatives of model outputs with respect to parameters using finite differences, the differences between incrementally-varied parameter values will be exactly correct because both PEST and the model use exactly the same parameter values.

The ability for PEST to compensate for limited parameter space widths on model input files is lost when parameter values are written to those files using program PAR2PAR (because PEST has no way of adjusting its internal representation of parameters based on PAR2PAR outputs). Thus unless the formatting requirements of the model input file are such that it allows model input parameters to be supplied with full numerical precision (which is normally about 7 significant figures), slight errors will be incurred in the derivatives calculation process. (Note that where a number is small or large enough for exponential notation to be required for its representation, up to 13 characters may be required for the

representation of that number using 7 significant figures.) Imprecision in derivatives calculation can have a profound effect on the outcome of an inversion process. *Thus, if the model permits it, you should make absolutely sure that the template files used by PAR2PAR to write model input files use parameter space widths which are as large as the model will tolerate (up to a maximum of the 13 characters if using single precision arithmetic, or 23 characters if using double precision arithmetic).* If model input file formatting requirements are too restrictive to allow a parameter value to be written without some loss of significance, then you should at least be aware of the fact that use of PAR2PAR under these circumstances has the potential to reduce the efficacy of PEST's performance.

10.7.3.3 Intermediate Files

Before it runs the model PEST deletes all model output files that it knows about (ie. the model output files cited in the PEST control file). Hence if the model fails to run, PEST will not read old model output files produced on previous model runs, mistaking them for new ones. Thus if PEST generates an error message saying that it cannot find a particular model output file, this is a sure sign that, for some reason, the model failed to run. In most cases the matter is then easily rectified by taking some simple measure such as altering the contents of the "model command line" section of the PEST control file.

Where a model is comprised of multiple executable programs listed in a batch file, similar considerations apply to "intermediate model files", ie. to files generated by one or more of the executable programs comprising the composite model and read by one or more succeeding executable programs cited in the model batch file. If, for some reason, an executable program which generates such an intermediate file fails to run, then later executable programs of the composite model may read old intermediate files, mistaking them for new ones. If this happens, model outputs will not reflect current parameter values; in fact, because they are independent of current parameter values, PEST will probably declare that at least some model outputs are insensitive with respect to some parameter values. This problem can be avoided if commands are included in the model batch file to delete all intermediate files before any of the executable programs comprising the model are run. If PAR2PAR is one such executable program, then all model input files cited in the "template and model input files" section of the PAR2PAR input file should be deleted prior to running PAR2PAR. Example 10.11 shows an example of a model batch file in which this precaution is taken.

```
rem Model input files written by PAR2PAR are deleted.
del model1.in
del model2.in
rem PAR2PAR is run.
par2par par2par.in
rem The model is run.
model
```

Example 10.11. A model batch file which includes PAR2PAR as one of the model executable programs.

In the batch file depicted in Example 10.11, file *par2par.in* is the PAR2PAR input file. If it is desired that screen output from all programs comprising the composite model (including the model batch file itself) be suppressed so that the model's screen output does not interfere

with that of PEST, the batch file shown in Example 10.11 could be altered to that shown in Example 10.12.

```
@echo off
rem Model input files written by PAR2PAR are deleted.
del model1.in > nul
del model2.in > nul
rem PAR2PAR is run.
par2par par2par.in > nul
rem The model is run.
model > nul
```

Example 10.12. The batch file of Example 10.11 with all screen output suppressed.

11. SENSAN

11.1 Introduction

In many modelling applications, an analysis of the sensitivity of particular model outputs to particular model inputs must be performed. 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 PEST, writing model input files on the basis of user-supplied templates, and reading output files with the aid of a user-prepared instruction set. In fact, SENSAN communicates only indirectly with a model, using the PEST utilities TEMPCHEK and INSCHEK to write and read model files; these programs are run by SENSAN as "system calls".

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.

11.2 SENSAN File Requirements

11.2.1 General

SENSAN requires four types of input file. The first two are the SENSAN control file and the 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.

11.2.2 Template Files

Section 3.2 of this manual provides a detailed discussion of how PEST writes parameter values to model input files.

After a user has prepared a template file prior to running PEST, he/she can check its integrity using the PEST utility TEMPCHEK. As explained in Chapter 10, 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. Rather than reproduce this functionality within SENSAN, SENSAN simply 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 in which the executable file tempchek.exe resides is either the current directory or is a directory 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 may 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.

11.2.3 Instruction Files

Instruction files are discussed in Section 3.3 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 Section 10.2. INSCHEK is also capable of actually reading values from a model output file on the basis of a user-supplied instruction file. In order to avoid duplication of this functionality, 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 reside either in the current directory or within a directory 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.

11.2.4 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

```
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
```

Example 11.1 A parameter variation file.

presented below.

The file shown in Example 11.1 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 the 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 below) 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.

11.2.5 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.

Example 11.2 shows a SENSAN control file. Example 11.3 shows the structure 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; viz. "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 are provided with identical names to the corresponding PEST variables.

```
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
```

Example 11.2 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)
```

Example 11.3 Structure of the SENSAN control file.

The role of each SENSAN input variable is now discussed.

11.2.6 Control Data

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 “nul” 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 the user should ensure that the model likewise produces no screen output by redirecting its output to the “nul” file using, for example, the command

```
model > nul
```

to run the model. If SENSAN is thus left to produce the only screen output, the user can monitor 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”, ie. 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

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

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 3.2.6 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 3.2.6 for more details.

11.2.7 SENSAN Files

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.

11.2.8 Model Command Line

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, model screen output can be redirected to the *nul* file. See Example 11.4.

```
@echo off
model1 > nul
model2 > nul
```

Example 11.4. A batch file serving as a model; all screen output has been disabled.

Care should be taken if SENSAN is executing in the "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.

11.2.9 Model Input/Output

TEMPFLE

TEMPFLE is the name of a template file used to write a model input file.

INFLE

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

The name of the model output file read by the instruction file whose name precedes it in the SENSAN control file.

11.2.10 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. Example 11.5 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
```

Example 11.5. A parameter variation file for a SENSAN run in which system commands are run after the model.

In Example 11.5 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 for a particular 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 the "noverbose" mode, for then the string "> nul" is added to any command appearing in the parameter variation file. This may cause problems if a command already uses the ">" symbol to redirect command output to a file.

11.3 SENSCHEK

11.3.1 About SENSCHEK

Once all SENSAN input data has been prepared, and before running SENSAN, SENSCHEK should be run in order to verify that the entire SENSAN input dataset is correct and consistent. SENSCHEK reads all SENSAN input files, ie. the SENSAN control file, the parameter variation file, as well as all template and instruction files. It checks all of these files for correct syntax and for consistency between them. Thus, for example, if the number of observations cited in all instruction files differs from the value supplied for NOBS in the SENSAN control file, or if values are provided for parameters in the parameter variation file which are not cited in any template file, SENSCHEK will detect the error and write an appropriate error message to the screen.

Though SENSAN itself carries out some error checking, it has not been programmed to carry out extensive consistency checks in the way that SENSCHEK does. In fact SENSAN may run happily if provided with certain erroneous datasets; however SENSAN's results under such conditions will be misleading. *Thus it is most important that SENSCHEK be run prior to SENSAN, once all SENSAN input files have been prepared.*

11.3.2 Running SENSCHEK

SENSCHEK is run using the command

```
senschek infile
```

where *infile* is the name of the SENSAN control file. If the latter possesses an extension of ".sns", then this extension can be omitted from the filename in the same manner that the ".pst" extension can be omitted from the name of the PEST control file when running PEST and PESTCHEK.

SENSCHEK writes its error messages to the screen. It is important to note that if SENSCHEK detects certain errors early in the SENSAN control file it may not proceed with its checking of the remainder of this file, nor of the template and instruction files cited in the SENSAN control file, nor of the parameter variation file. Thus it is important to ensure that once a SENCHEK-identified error has been rectified, SENSCHEK is run again. Only when SENSCHEK explicitly informs the user that no errors have been detected in the entire SENSAN input dataset is it safe to run SENSAN.

11.4 Running SENSAN

11.4.1 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 directory, or are in a directory 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.

11.4.2 Interrupting SENSAN Execution

To interrupt SENSAN type <Ctl-C>.

11.5 Files Written by SENSAN

11.5.1 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 (ie. "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 (ie. 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:

$$\frac{O_p - O_b}{O_b} \tag{11.1}$$

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, ie. 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:

$$\frac{O - O_b}{p - p_b} \tag{11.2}$$

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 the Jacobian matrix.

Note that if $p - p_b$ in equation 11.2 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 11.2 if $p < p_b$.*

11.5.2 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 filename should be avoided when naming other files.

INSCHEK writes the values of the observations which it reads from a model output file to 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.

11.6 Sensitivity of the Objective Function

SENSAN allows a user 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 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 Example 11.6.

```
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
```

Example 11.6 A SENSAN control file with PEST as the model.

Note the following points:-

- There can be as many parameters as you like but only one observation. This should be the initial value of phi as read from the PEST run record file; PEST writes this value after it has carried out just one model run.
- 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.
- Use the single window version of PEST rather than Parallel PEST.

The instruction set by which the PEST control file is 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.

11.7 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 possible that model execution will fail for some parameter value sets supplied by the user. 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, 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" (ie. an INSCHEK-generated file) is that it was unable to run INSCHEK and/or TEMPCHEK because their directories 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 *nul* 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 (ie. *t###.par* written by SENSAN) that contains the values of more parameters than are cited on any one template file.

11.8 An Example

Included in the *pestex* subdirectory of the directory into which PEST was installed are the files required to run the soil clod shrinkage example discussed in Chapter 12 of this manual. Also included in this subdirectory are three files not discussed in Chapter 12. These are *twofit.sns* a SENSAN control file, *out1.ins* an instruction file identical to *out.ins* discussed in Chapter 12, 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 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 12) 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 until later inspection.

Before running SENSAN make sure that the PEST directory is cited in the PATH environment variable (so that SENSAN can run TEMPCHEK and INSCHEK). Run SENSCHEK using the command:

```
senschek twofit
```

After 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.

12. An Example

12.1 Parameter Estimation

12.1.1 Laboratory Data

This section takes you, step by step, through an 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* subdirectory of the main PEST directory.

Table 12.1 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 12.1; 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 12.1.)

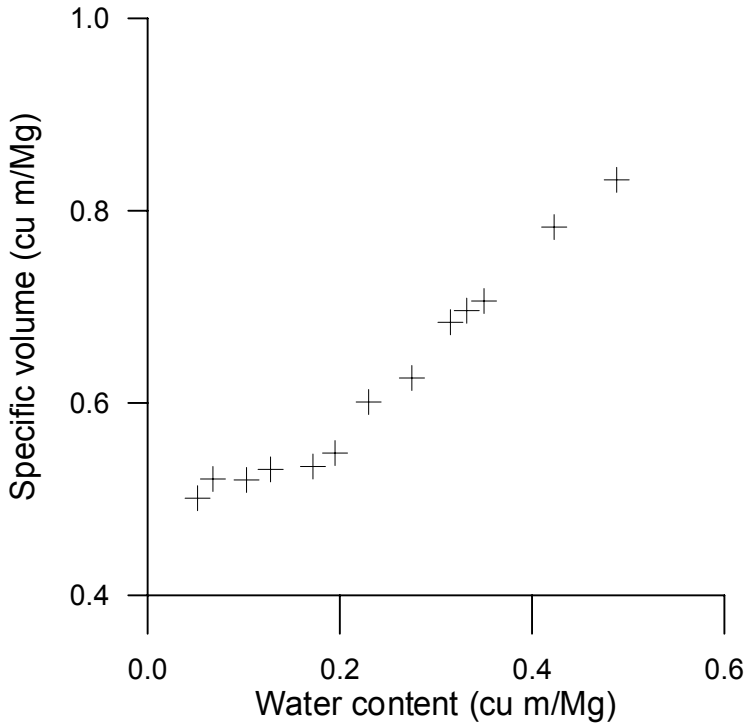


Figure 12.1 Soil clod shrinkage data.

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 12.1 Soil clod shrinkage data.

12.1.2 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 12.2 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

$$\begin{aligned} y &= s_1 x + y_1 && x \le x_c \\ y &= s_2 x + (s_1 - s_2) x_c + y_1 && x > x_c \end{aligned} \qquad (12.1)$$

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 Example 12.1 (see also file *twoline.for* in the *pestex* subdirectory). 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 the water contents (ie. x values in equation 12.1) 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. Example 12.2 shows a typical TWOLINE input file, while Example 12.3 shows a corresponding TWOLINE

output file. An executable version of TWOLINE (viz. *twoline.exe*) is provided in the *pestex* subdirectory of the PEST directory.

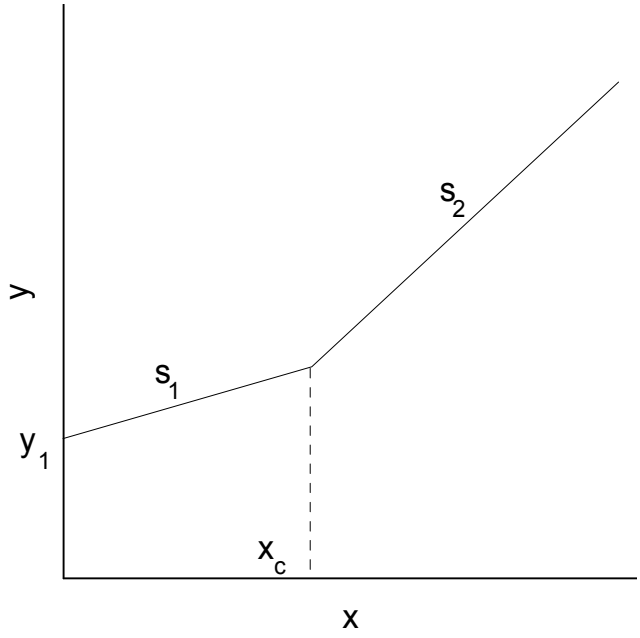


Figure 12.2 Parameters of the two line model.

```
      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
```

Example 12.1 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 12.1. The input file of Example 12.2 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, viz. 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
```

Example 12.2 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
```

Example 12.3 A TWOLINE output file *out.dat*

12.1.3 Preparing the Template File

First a template file must be prepared. This is easily accomplished by copying the file *in.dat* listed in Example 12.2 to the file *in.tpl* and modifying this latter file in order to turn it into a PEST template file. Example 12.4 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 3.2.5, 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
```

Example 12.4 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
```

Example 12.5 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. Example 12.6 shows such a file; because this file will shortly be used with program PESTGEN to generate a PEST control file, the values supplied for each of the parameters are the initial parameter values to be used in the optimisation process.

```
 s1
 s2
 y1
 xc
```

Example 12.5 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
```

Example 12.6 File *in.par*

At this stage TEMPCHEK should be run again using the command

```
tempchek in.tpl in.dat in.par
```

("*in.par*" can be omitted if you wish, for this is the default parameter value filename generated automatically by TEMPCHEK from the template filename.) When invoked with 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.

12.1.4 Preparing the Instruction File

Next the instruction file should be prepared. This can be easily accomplished by writing the instructions shown in Example 12.7 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
```

Example 12.7 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 Example 12.8.

```
 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
```

Example 12.8 File *out.obf*

12.1.5 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 optimisation 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 12.1, ie. with the appropriate laboratory measurement; see Example 12.9. 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
```

Example 12.9 File *measure.obf*

PESTGEN generates a PEST control file named *twofit.pst*; see Example 12.10. 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 (Example 12.11 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
```

Example 12.10 The PESTGEN-generated control file *twofit.pst*

```
* model command line
twoline
* model input/output
in.tpl  in.dat
out.ins  out.dat
```

Example 12.11 Altered section 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* will be written by PEST in the *pestex* subdirectory; so too will file *twofit.par* containing the optimised parameter set. Figure 12.3 shows the lines of best fit superimposed on the laboratory data.

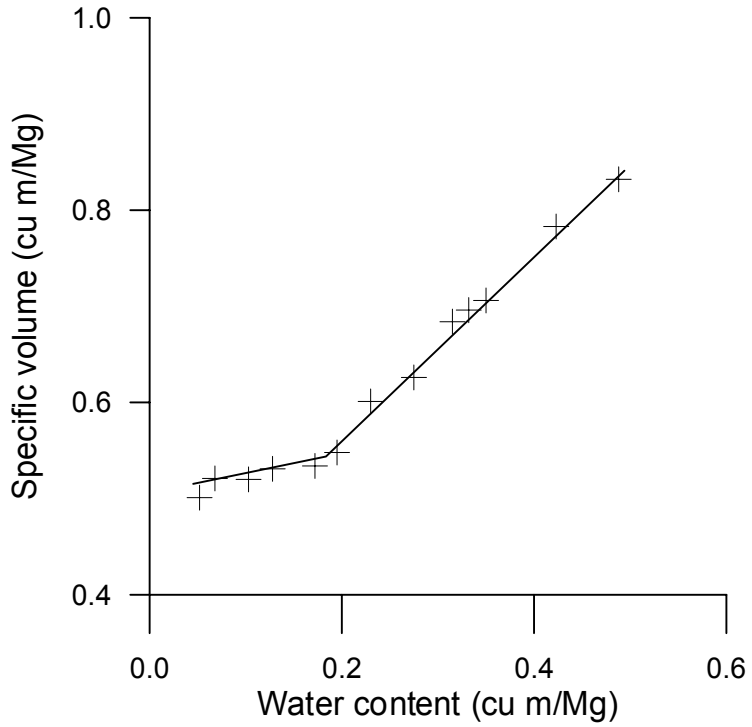


Figure 12.3 Soil clod shrinkage data with lines of best fit superimposed.

12.2 Predictive Analysis

12.2.1 Obtaining the Model Prediction of Maximum Likelihood

Files for this example can be found in the *\papestex* directory of the PEST directory after installation. This example builds on the soil clod shrinkage example discussed in the previous section.

Based on the dataset supplied with the 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 12.2.

Parameter	PEST-estimated value
s1	0.238
s2	0.963
y1	0.497
xc	0.174

Table 12.2. Optimised parameter values for soil clod shrinkage example.

After PEST is run in parameter 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 with 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 were 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.

12.2.2 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 supplied. Example 12.12 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
```

Example 12.12. A batch file encompassing a 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 *nul* file instead of the screen. To find out more about batch files, see the DOS help file on your Windows CD.

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.)

12.2.3 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 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 (ie. NOBS) must be increased from "13" to "14" on the 4th line of file *twofit1.pst*.

3. There will now be two observation groups, so alter the 5th entry on line 4 of file *twofit1.pst* (ie. NOBSGP) to "2".

4. There are now two model input files and two model output files, so alter the first two entries on the 5th line of file *twofit1.pst* (ie. 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 are ignored by PEST when run in predictive analysis mode. 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 parameter 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 (ie. *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 underneath 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 (ie. *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, ie. 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 6, 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.

12.2.4 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).

12.2.5 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 (ie. 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 is 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 12.4 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.

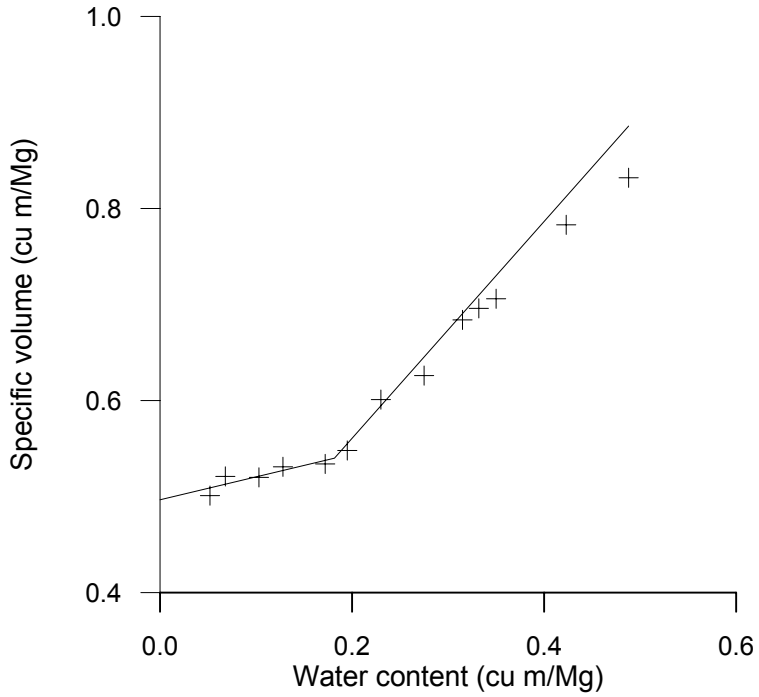


Figure 12.4. Soil clod shrinkage data with line segments superimposed.

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.

13. Frequently Asked Questions

13.1 PEST

When I run PEST with certain older models, a message pops up asking me if I would like to run the model in MS-DOS mode. How can I prevent this from happening?

Click with your right mouse button on the bar at the top of the DOS window in which you are running PEST and choose "properties" from the pop-up menu. In the "program/advanced" section of the pop-up dialogue box uncheck the "Suggest MS-DOS mode as necessary" box. To prevent this from ever happening again, click on the model icon in Windows Explorer with the right mouse button and alter its properties in similar fashion.

When I run PEST with a certain model a message pops up telling me that the computer is about to enter MS-DOS mode and that I must close all other programs. How can I prevent this from happening?

Click on the model icon in Windows Explorer as described above. This time uncheck the "MS-DOS" mode box.

I am having trouble running PEST in WINDOWS NT and/or WINDOWS 2000. Sometimes a command line window just disappears.

See Section 13.3 below.

13.2 Parallel PEST

Can a slave be introduced part of the way through a Parallel PEST run?

As presently programmed, no. All incidences of PSLAVE must begin execution before PEST. A slave can drop out of the Parallel PEST optimisation process, or its execution can be terminated (eg. by hitting <Ctl-C>) and PEST will simply carry on without it as long as there are other slaves to carry out model runs. However if this slave is then re-started, PEST will not recognise it. Even if you start a slave just after PEST has started, before it has actually undertaken any model runs, PEST will still not recognise it as it must be started before PEST commences execution.

Our office network is such that we have a single server to which all of our individual machines have access as drive O. Our machines are not able to communicate individually with each other in a peer-to-peer sense. What is the best way to set up Parallel PEST under these circumstances?

Install the model on each machine. Create as many subdirectories on drive O: as there are slaves, calling these subdirectories, for example, *sub1, sub2*, etc. On each slave machine make one such directory the current working directory, making sure to use a different subdirectory in each case. Make sure the model can run from that subdirectory once PEST-generated input files are placed there (ie. do a "dummy" model run on each slave machine

using normal model input files placed in each of the *sub1, sub2*, etc. subdirectories). Then start up PSLAVE on each slave machine (one of which will also probably be the machine running PEST).

The PEST template and instruction files can reside in any directory on the master machine. However for each slave, Parallel PEST should be informed that model input files and model output files reside in the *sub1, sub2* etc. subdirectory appropriate to that slave. Each such subdirectory should also be cited as a slave subdirectory for the purposes of writing the signal files listed in Table 9.1. Once the Parallel PEST run management file has been constructed accordingly, Parallel PEST can be started and should run without problems.

If the slave and model working directories are on a server's disk, is not the server running the model?

No, the model is run by the machine from which PSLAVE is launched irrespective of whether the current working directory is on the current machine's disk, or on a disk belonging to another machine. Similarly, even if the model executable resides on another machine's disk, the model is run by the machine from which the model run command is issued; this is the machine running PSLAVE, for it is PSLAVE which issues the command to run the model. If the model executable resides on another machine, then it has to be loaded into the current machine's memory across the network prior to execution.

Do I have to install the model executable on each slave machine?

Not at all. If a single model executable is accessible through the network by all slave machines then it can be run from each slave machine without having to install it on any of them. The disadvantage of this scheme, however, is that if the model executable is large, reloading it to each slave machine every time that slave runs the model may be a slow process on heavily-used networks. The contribution that this makes to network traffic may result in a user having to employ a higher value for WAIT than would otherwise be the case, further slowing down Parallel PEST's operations. However with disk caching, these problems may be mitigated somewhat.

When I came to work one morning after Parallel PEST had been running all night, one of my slaves had ceased carrying out model runs, displaying instead the following prompt:

```
Sharing violation reading drive C
Abort, Retry, Fail?
```

Why did this happen?

This message was sent by the operating system, not by PSLAVE. The model was probably trying to read an input file that had not yet been closed by PEST, or PEST was trying to read a model output file that had not yet been closed by the model. Theoretically, such things should not happen. However on busy machines connected to busy networks it may take some time for one process on a particular machine to get the message that a file has been closed by a process running on another machine. To prevent this error from happening increase the value of WAIT, the length (in seconds) of the pause made by PSLAVE and PEST at certain strategic stages of run management and data exchange.

Can PSLAVE run a batch file containing multiple executables as the model, just like PEST

can?

Yes. However make sure that either the full directory of each executable is included within its name as cited in the batch file, or that the directory containing each executable is cited in the PATH environment variable on the machines from which the model is run.

Can PEST and PPEST be used interchangeably?

Yes, all versions of PEST use identical template and instruction files, and an identical PEST control file.

Parallel PEST terminated execution with an error message to the effect that it could not find the model output file. What could be wrong? Furthermore, it waited an unduly long time after commencement of execution to inform me of this.

If Parallel PEST encounters any problems whatsoever in reading a model output file (for example premature termination of the file, data errors, poor instructions or file not present), it tries to read the file three times at 30 second intervals in order to be sure that the problems are not network-related. If, after making three attempts to read the file, it is still unsuccessful, it reports the error to the screen and terminates execution.

If a model output file cannot be found then either the model did not write it, the model did not run, or an erroneous name was provided for the model output file in the Parallel PEST run management file. In the first case, a poor parameter set may have been provided to the model, resulting in a model run-time error being generated before the model wrote any data to its output file. Or perhaps model input files were misnamed in the Parallel PEST run management file. Alternatively, if the model did not run, the model command line, as provided to PSLAVE, may have been inappropriate. To find out more about the cause of the problem, ascertain the name of the slave where the trouble occurred (this should be apparent from the PEST error message issued on termination of its execution); then shut down that slave and run the model from the slave working directory using the same command as that used by PSLAVE and watch what happens. But before doing this, check that the model output files are not, in fact present after all. If they are, then they must have been misnamed in the Parallel PEST run management file.

If, after following the steps outlined above, the reason for the problem is still not apparent, increase the value of the WAIT variable and try running Parallel PEST again.

13.3 PEST and Windows NT

A few problems have been encountered when using PEST and Parallel PEST with WINDOWS NT and WINDOWS 2000. The problems are not caused by PEST. However this is not much consolation if PEST does not perform correctly.

Problems encountered include the following.

- A command line window used by PEST or PSLAVE closes in the middle of a model run.

- PEST reports that it cannot find a model output file when the model output file is actually present.

- In a composite model written as a batch file and containing a number of executables run in succession, one of the executable programs is not actually executed on one particular model run.

The problems are particularly frustrating because they occur so rarely. Thus PEST may run happily for hours when, suddenly, one of the above conditions occurs. Any one of them will cause immediate termination of execution.

In many cases where these problems have been encountered it has been found that the problem was rectified by re-compiling the model source code using a modern compiler. Old compilers used DOS memory extenders (some of which had a memory leakage bug) to gain access to all of a machine's memory; very old compilers generated 16 bit executables and used no extender at all. WINDOWS NT/2000 does not work well with either of these.

If the problem persists, make sure that the latest service pack has been installed. Also, see if use of the *command* rather than the *cmd* command interpreter (and vice versa) makes any difference.

Index