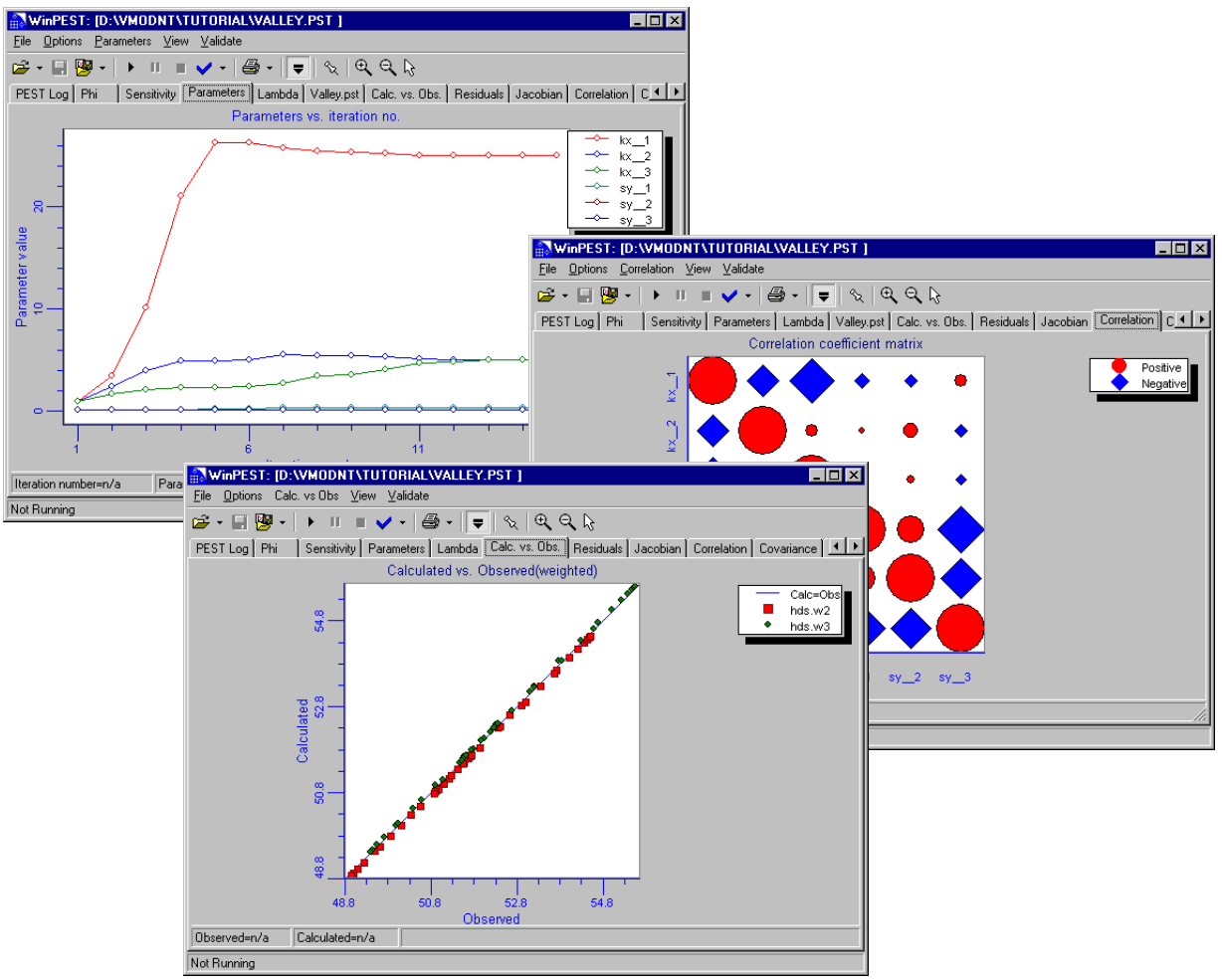
USER'S MANUAL

for

WinPEST

The Windows® interface to PEST - the popular parameter estimation program for professional groundwater modelling using Visual MODFLOW.



DISCLAIMER OF WARRANTY

This manual and associated software are sold "as is" and without warranties as to performance or merchantability. The seller's salespersons may have made statements about this software. Any such statements do not constitute warranties.

This program is sold without any express or implied warranties whatsoever. No warranty of fitness for a particular purpose is offered. The user is advised to test the program thoroughly before relying on it. The user assumes the entire risk of using the program. Any liability of seller or manufacturer will be limited exclusively to replacement of diskettes defective in materials or workmanship.

Waterloo Hydrogeologic Inc.
180 Columbia Street West - Unit 1104
Waterloo, Ontario, CANADA
N2L 3L3

Phone (519) 746 1798
Fax (519) 885 5262

Email: techsupport@flowpath.com
Web: www.flowpath.com

Table of Contents

1

1 - Introduction to PEST

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. Modeling programs generally require the following four types of data, although the distinction between them may not always be clear:

- **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 wells.
- **Parameters.** These are the properties of the system. Parameters for a ground water model include the hydraulic conductivity and storage capacity of the porous media through which the water flows. A model may have many parameters. Each pertaining to one particular attribute of the system which affects the model's response to an input or disturbance. In spatial models a system property may vary from place to place. Hence the parameter data needed by the model may include either individual values of a property for certain model subregions, or values which describe the manner in which the property is spatially distributed.
- **Disturbances**. These are the quantities which "drive" the system, for example recharge data in a groundwater model, and the source and location of contaminants. Like parameters, disturbances may be spatially dependent.
- **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 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 disturbances. 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 solving a set of nonlinear partial differential equations.

Where a model simulates reality, often the model-user does not know what the reality is. In fact, models are often used to infer reality. For example, 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 to achieve this match are likely correct. This is fortunate, as it is often difficult or expensive to measure hydraulic conductivity directly.

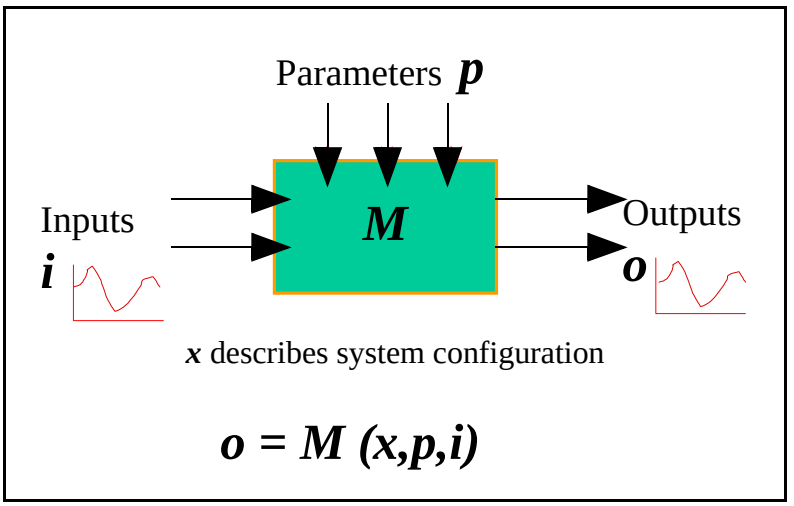


Figure 1.1: Typical model structure.

What PEST Does

PEST is all about using existing models to infer aspects of reality that may not be amenable to direct measurement. In general, its use falls into the following two broad categories:

- **Interpretation**. An experiment is often set up to specifically infer some property of a system, often by disturbing the system in some way (e.g. a pumping test). The model is then used to relate the disturbance and system properties to values that can be measured (e.g. piezometer measurements). The measured data may then be interpreted based on the premise that, for a known disturbance, it is possible to estimate the system properties from the measurement set (e.g. hydraulic conductivity from piezometer data)
- **Calibration**. If a system is disturbed, and this disturbance is simulated in a model, it should be possible to adjust the model's parameters until the model output corresponds to field measurements taken during the disturbance. If so, we often conclude that the model will represent the system's behavior in response to other disturbances - disturbances which we may not be prepared to

do in practice. A model is said to be "calibrated" when its parameters have been adjusted in this fashion.

The purpose of PEST is to assist in data interpretation and in model calibration. PEST will adjust model parameters and disturbances (hereafter referred to only as parameters) until the fit between model outputs and laboratory or field observations is optimized. While this is nothing new, the usefulness of PEST lies in its ability to perform this optimization for **any** model that reads its input data from one or more ASCII (i.e. 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 for a wide range of model types. This model-independence makes PEST unique. PEST can turn just about any existing computer model into a powerful nonlinear estimation package, be it a homemade model based on an analytical solution to a simple physical problem, or a sophisticated numerical solver for a complex boundary-value problem.

Models produce numbers. If there are field or laboratory measurements corresponding to some of these numbers, PEST can adjust model parameters such that the discrepancies between the pertinent model-generated numbers and the corresponding measurements are minimized. It does this by running the model as many times as is necessary to determine this optimal set of parameters. You, as the model user, must tell PEST what the adjustable parameters are. Once PEST is provided with this information, it can rewrite the model-input files using whatever parameters are appropriate at any stage of the optimization process. You must also tell PEST what model output values correspond to your observations. Thus, each time it runs the model, PEST is able to read the model outcomes that correspond 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 the model-input data and runs the model again.

However, for PEST to take control of an existing model and optimize its parameters the following conditions must be met:

- The input files containing the parameters that PEST is required to adjust must be in ASCII (i.e. text) format.
- The output files containing the model outcomes that complement field or laboratory measurements must be in ASCII (i.e. text) format.
- The model must be able to run from a typed command line and must not require user intervention during the run (see below for further details).
- The Gauss-Marquardt-Levenberg nonlinear estimation technique used in PEST requires that the output values generated by the model, which correspond to the observations, must change smoothly and continuously for all input parameter values. That is the relationship between the input parameters and the output

"observations" must be continuously differentiable.

How PEST Works

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

- parameter definition and recognition,
- observation definition and recognition, and
- parameter estimation algorithm.

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

Parameter Definition and Recognition

Of the masses of data that may be in a model's input files, those numbers must be identified which PEST is free to alter and optimize. 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 these two input files. Visual MODFLOW constructs the necessary template files depending on the parameters that you chose. Then whenever PEST runs the model it copies the template to the model input file, putting the proper parameter value into the template as it does so.

With respect to the parameter template files the following points are noteworthy:

- During a PEST run a parameter can remain fixed if desired. Thus, while the parameter may be identified in the template file, PEST will not adjust its value from the value you supply at the beginning of the parameter estimation process.
- One or a number of parameters can be "tied" to a "parent" parameter. In this case, only the parent parameter is actually optimized and 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 all parameters that are neither fixed nor tied. This information is vital to PEST, for it informs PEST of the range of permissible values that a parameter can take. For example, parameters such as hydraulic conductivity and solute concentration should never be have negative values.
- For many models it has been found that the amount of time needed to find an optimum set of parameters can be greatly reduced if the logarithms of certain parameters are optimized, rather than the parameters themselves.
- Finally, parameters adjusted by PEST can be scaled and offset. Thus you may wish to subtract 273.15 from an absolute temperature before writing that temperature to a model input file, which requires Celsius degrees.

Observation Definition and Recognition

Of the masses of data produced by a model, only a handful of numbers may actually correspond to "observations". For example, a groundwater model may calculate head values at thousands of nodes of a finite-difference grid, however, head measurements may be available at only a handful of piezometers. PEST must be able to identify a handful of numbers out of the thousands that may be written to the model's output file. Unfortunately, the template concept used for model input files will not work for model output files since model output files 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. All PEST requires is an instruction file be provided detailing how to find those observations.

Once interfaced with a model, PEST's role is to minimize 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". Weighting these discrepancies allows you to make some observations more important than others. Weights should be inversely proportional to the standard deviations of observations. "Trustworthy" observations having a greater weight than those that can be less trusted. Also, if observations are of different types (e.g. head measurements and stream baseflo values) the weights assigned to each type should reflect the relative magnitude of the quantities. In this way, larger numbers will not dominate the parameter estimation process just because the numbers are large. An observation can be provided with a weight of zero if you do not wish it to affect the optimization process at all.

The Parameter Estimation Algorithm

The Gauss-Marquardt-Levenberg algorithm used by PEST is described in detail in the next chapter. However, a summary of the parameter estimation process is provided here.

For linear models (i.e. models for which observations are calculated from parameters through a matrix equation with constant parameter coefficients), optimization can be achieved in one step. However for non-linear 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 series expansion about the current 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 the changes in parameters to the improvement in the objective function, PEST can tell whether it is worth doing another optimization 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 optimization iteration. For

many problems only five or six optimization iterations will be required for model calibration or data interpretation. In other cases, convergence will be much slower Often the proper choice of whether and what parameters should be logarithmically transformed can have a pronounced effect on the optimization 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 optimization 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 to calculate their derivatives with respect to the parameter. This is repeated for each parameter. This technique of derivative 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 round-off errors will detract from derivatives accuracy. Both of these effects will degrade optimization performance. To combat such 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 optimization 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.

In the course of the estimation process PEST writes what it is doing to the screen. PEST simultaneously writes a more detailed run record to a file.You can stop PEST execution at any time and recommence execution exactly where it was interrupted. Alternatively, you can shut down PEST completely at any stage and restart it later at either the beginning of the optimization 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 observations. If PEST’s performance is being hindered by the behavior of certain parameters (normally the most insensitive ones), these parameters can be temporarily held at their current values while PEST calculates a suitable upgrade vector for the rest of the parameters. If desired, PEST can be requested to repeat its determination of the parameter upgrade vector with additional parameters held fixed.Variables governing

the operation of the Gauss-Marquardt-Levenberg method in determining the optimu 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.

At the end of the parameter estimation process (the end being determined either by PEST or by you) PEST records the optimized value of each adjustable parameter together with its 95% confidence interval. It tabulates the set of field measurements, their optimized model-calculated counterparts, and the difference between each pair. Then it calculates and prints the parameter covariance matrix, the parameter correlation coefficient matrix and the matrix of normalized eigenvectors of the covariance matrix.

2

2 - The Mathematics of PEST

Parameter Estimation for Linear Models

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

$$Xb = c \qquad (2.1)$$

In equation (2.1) **X** is a $m \times n$ matrix, i.e. 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 that 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 disturbances 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 disturbances 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 syste parameters.)

Most models generate a wealth of data for which we usually only have a handful of corresponding field measurements. Therefore, we will use the word "observations" to describe the elements of the vector **c** even though the model in fact, generates **c**. As we include in the vector **c** only those model outcomes for which there are complementary 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

$$x_{p1}b_1 + x_{p2}b_2 \ldots x_{pn}b_n = c_p \qquad (2.2)$$

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*, that is we are capable of observing the syste 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, i.e. elements of **b**?" If **b** were 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* should 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 this number is (referred to as the "objective function") the greater is the consistency between model and observations and the greater is our confidence that the determined parameter set is the correct one. Expressing this mathematically, we wish to minimize Φ, where Φ is defined by the equation

$$\Phi = (c - Xb)^t (c - Xb) \qquad (2.3)$$

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

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

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 realization 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 **c** are but one particular realization**.** 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

$$\sigma^2 = \frac{\Phi}{(m-n)} \qquad (2.5)$$

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

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

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

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} \qquad (2.7)$$

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 normalized eigenvectors of the covariance matrix C(***b***). If each eigenvector is dominated by one element, individual parameter values are 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.

Adding Observation Weights

The discussion so far presupposes that all observations are equally weighted in the parameter estimation process. However this will not always be the case as some measurements may be more uncertain than others.

Another problem arises where observations are of more than one type. For example, you may have a set of head measurements at several piezometers and a couple of strea baseflow measurements. However, the units for these two quantities are different (m and m^3/s respectively) and hence the numbers used to represent them may be of vastly different magnitudes. Under these circumstances the quantity with the numerically larger value will dominate 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 values 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:

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

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. Equation (2.8a) is equivalent to:

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

Where,

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

C(***c***) represents the covariance matrix of the *m*-dimensional observation random vector ***c*** of which our measurement vector **c** is a particular realization. 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

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

While equation (2.6) for the parameter covariance matrix becomes,

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

Using Prior Information to Improve Parameter Estimation Process

Often some independent information exists about the parameters that we wish to optimize. This information may be in the form of unrelated estimates or of relationships between parameters expressed in the form of equation (2.2). When this information is included, it can lend stability to the parameter estimation process, especially when parameters are highly correlated. Correlated parameters can lead to non-unique parameter estimates because varying them in certain linear combinations may cause very little change in the objective function. In some cases, this non-uniqueness 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 non-uniqueness and provide stability

Parameter estimates will also be non-unique if there are less observations than parameters. Equation (2.11) is not solvable under these conditions as the matrix $\mathbf{X^tQX}$ 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 row 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. In theory, this weight should be 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, however, the user simply assigns the weights according to 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 a "penalty function". The aim of the estimation process is to lower the objective function defined by equation (2.9) 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 difference between model outcomes and field measurements. However, when prior information is included, a "penalty" equal to the square of the difference between what the right hand side of the prior information equation should be, and what it currently is, is introduced into the objective function. This difference is multiplied by the square of its weight before including it in the objective function.

Extending Linear Parameter Estimation to Non-Linear Models

Most models are non-linear, i.e. the relationships between parameters and observations are not of the type expressed by equations (2.1) and (2.2). Non-linear models must be "linearized" for the theory presented so far to be used in the estimation of their parameters.

To “linearize” a non-linear model, 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 that will become apparent, 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 $\mathbf{c}_0$, i.e.

$$c_0 = M(b_0) \qquad (2.13)$$

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

$$c = c_0 + J(b - b_0) \qquad (2.14)$$

Where **J** is the Jacobian matrix of *M*, i.e. the matrix composed 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 linearization 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, i.e. we wish to determine a parameter set for which the objective function Φ defined by

$$\Phi = (c - c_0 - J(b - b_0))^t Q(c - c_0 - J(b - b_0)) \qquad (2.15)$$

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 (**c** - $\mathbf{c}_0$) of equation (2.15) and **b** from equation (2.8a) is replaced by (**b** - $\mathbf{b}_0$) from equation (2.15). Thus we can use the theory for linear parameter estimation to calculate the parameter upgrade vector (**b** - $\mathbf{b}_0$) on the basis of the vector (**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

$$u = (J^t Q J)^{-1} J^t Q(c - c_0) \qquad (2.16)$$

And equation (2.12) for the parameter covariance matrix becomes,

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

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

Since 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 optimization 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 non-linear 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 good to supply an initial parameter set $\mathbf{b}_0$, which approximates the true parameter set. A suitable choice for the initial parameter set can also reduce the number of iterations necessary to minimize 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 optimization stability and may reduce the number of iterations required to determine the optimal parameter set.

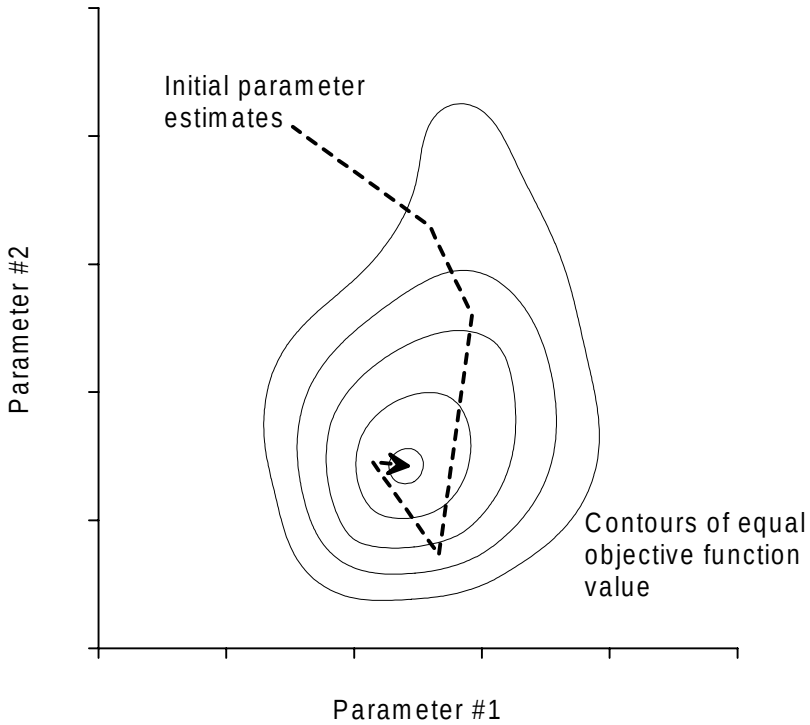


Figure 2.1: Iterative improvement of initial parameter values toward the global objective function minimum.

The Marquardt Parameter

Equation (2.16) forms the basis of non-linear weighted least-squares parameter estimation. It can be rewritten as

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

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

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

i.e. 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.

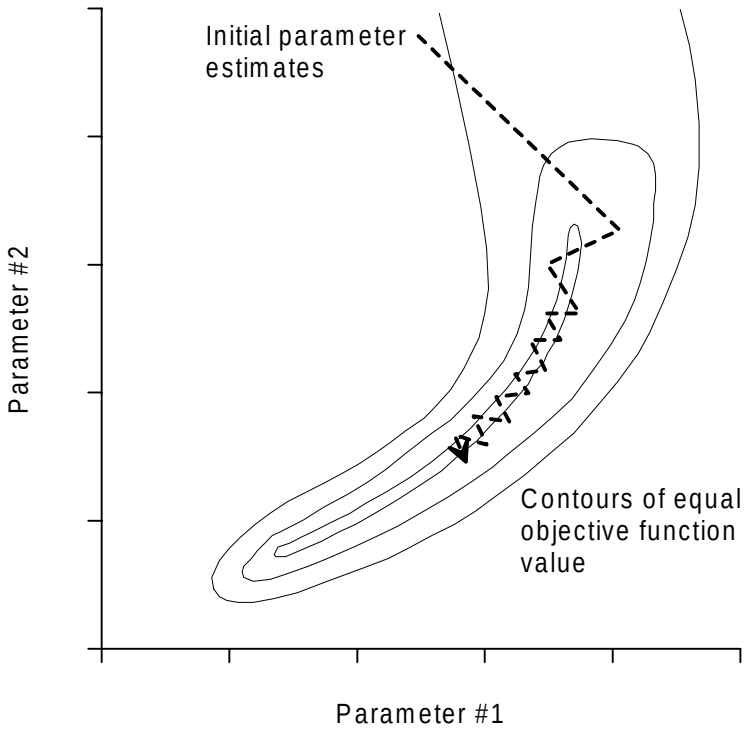


Figure 2.2: The phenomenon of "hemstitching".

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

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

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

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

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 optimization 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.

Parameter 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 round-off 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

$$s_{ii} = (J^t Q J)_{ii}^{-\frac{1}{2}} \qquad (2.22)$$

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

$$S^{-1}u = ((JS)^t QJS + \alpha S^t S)^{-1}(JS)^t Qr \qquad (2.23)$$

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 $(\mathbf{JS})^t\mathbf{QJS} + \alpha\mathbf{S}^t\mathbf{S}$ has all its diagonal elements equal to unity. For a non-zero α the diagonal elements of $(\mathbf{JS})^t\mathbf{QJS} + \alpha\mathbf{S}^t\mathbf{S}$ 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 $(\mathbf{JS})^t\mathbf{QJS} + \alpha\mathbf{S}^t\mathbf{S}$ of equation (2.23) will be $1 + \lambda$.

The Marquardt Lambda

As outlined at the end of the previous section, the largest element of $\alpha\mathbf{S}^t\mathbf{S}$ is denoted as λ and referred to as the Marquardt lambda. PEST requires that the user supply an initial value for λ. During the first optimization 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 obtained through solution of equation (2.23), and a new Φ 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 optimization iteration. Normally between one and four λ's need to be tested in this manner per optimization 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 the vast majority of 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 optimization iteration. However, this process makes PEST very "robust". If the optimization procedure slows down, changing λ in this fashion often gets the process moving again.

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 favorable, 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 β**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:

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

That is

$$\gamma = Ju \qquad (2.25b)$$

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

$$b = b_0 + \beta u \qquad (2.26)$$

3

3 - PEST's Implementation of the Method

The previous chapter discussed the theory behind PEST, that is the method of weighted least squares and its application to non-linear parameter estimation. This chapter discusses the way in which the least squares method has been implemented in PEST to provide a general, robust, parameter estimation package that is usable across a wide range of model types. Appendix B contains a detailed description of all the PEST control files and the parameters found in the control files.

Explanation of Parameter Operations

There are a number of parameter operations which can be performed by the user to increase the accuracy of any WinPEST run. The operations are as follows and are included in the following sections.

- Parameter Transformation
- Fixed and Tied Parameters
- Upper and Lower Parameter Bounds
- Scale and Offset
- Parameter Change Limits
- Damping of Parameter Changes
- Temporary Holding of Insensitive Parameters
- Observation Groups
- Termination Criteria

Parameter Transformation

PEST allows for the logarithmic transformation of some or all parameters. Often the parameter estimation process is much faster and more stable when PEST is asked to estimate the log of a parameter, rather than the parameter itself.

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

in the next section. 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 that has 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.

- PEST allows you to logarithmically transform parameters, which may improve the parameter estimation process.
- The co-variance, correlation coefficients and eigenvector values refer to the log of the parameter.
- However, the parameter estimates and confidence intervals refer to the untransformed parameter.
- Typically, parameters are log-transformed when their values can vary over several orders of magnitude (e.g. conductivity).
- The transformation of a parameter is defined by **PARTRANS** in the PEST control file (*projectname.pst*).

Fixed and Tied Parameters

PEST allows a parameter to be declared as "fixed" and take no part in the parameter estimation process. In this case, its value will not vary from its initial value. PEST also allows one or more parameters to be tied (i.e. linked) to a "parent" parameter. PEST does not estimate a value for a tied parameter. Rather PEST adjusts the parameter during the estimation process, such that the initial ratio to the parent parameter is maintained. 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 allows you to fix a parameter, which means it will not be part of the estimation process.
- PEST allows you to tie a parameter to another parameter.
- The ratio of a tied parameter to its parent remains constant during the estimation process.
- Parameters cannot be tied to other tied parameters or to fixed parameters.
- Whether a parameter is fixed or tied is defined by **PARTRANS** in the PEST control file (*projectname.pst*).

Upper and Lower Parameter Bounds

As well as supplying an initial estimate for each parameter, you must also supply parameter upper and lower bounds. These bounds define the maximum and minimum values, which a parameter is allowed to assume during the optimization process.

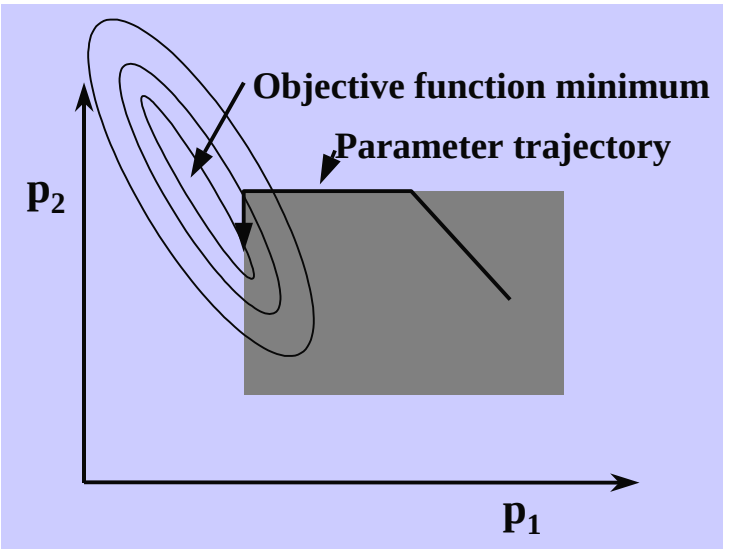


Figure 3.3: Example parameter trajectory for a two parameter model

It is important that upper and lower parameter bounds be chosen wisely. Often parameters can lie only within certain well-defined limits. For example, if the logarithm or square root of a particular parameter is taken during a simulation, then that parameter must never become negative or 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 adjust some parameters to extremely large or extremely small values (especially if the measured values are not consistent). Such extremely large or small values may result in floating point errors or difficulties with numerical convergence. Carefully choosing parameter bounds may circumvent this problem.

Figure 3.1 illustrates both the means that PEST uses for finding the minimum when parameter bounds are defined and the drawback to specifying improper bounds. For example, if a parameter upgrade vector is calculated which would cause a parameter to move beyond its bounds, PEST will instead assign the upper or lower bound to the parameter value. On the next iteration, if the upgrade vector would still take the parameter outside of the current bounds, PEST temporarily fixes the parameter. Such a

process is repeated for all the parameters until an upgrade vector is determined that either moves parameters from their bounds back into the allowed parameter domain, or leaves them fixed.

The strength of this strategy is that PEST can search along the boundaries of the parameter domain looking for the smallest value of the objective function, even though the global minimum of the objective function may lie outside of the parameter domain.

The obvious drawback of setting bounds is that the global minimum might lie outside of the bounds that you set. Therefore, it is important to chose your bounds appropriately.

At the beginning of each new optimization iteration all temporarily-frozen parameters are freed to allow them to move back inside the allowed parameter domain. The stepwise, temporary freezing of parameters is then repeated.

- It is important to chose upper and lower bounds wisely.
- If an updated parameter value is outside of its bounds, PEST temporarily holds the parameter at its boundary value.
- The strategy that PEST uses, allows PEST to search along the bounds of the parameter domain looking for the minimum value of the objective function
- A parameter's upper and lower bounds are defined by **PARLBND** and **PARUBND** in the PEST control file (*projectname.pst*).

Scale and Offset

Before writing a parameter value to a model input file, PEST multiplies the value by the scale and adds the offset. Both of which must be specified for every parameter.

The scale and offset variables can be very convenient in some situations. For example, for a parameter, such as elevation, you may wish to redefine the parameter that PEST optimizes as the elevation minus some datum. In this case, the result may be thickness, which may be a more "natural" parameter for PEST to optimize than elevation. In particular, it may make more sense to express a derivative increment as a fraction of the thickness rather than as a fraction of the elevation. Also, the optimization process may be better behaved if the thickness parameter is log-transformed. Again it would be surprising if the log-transformation of elevation improved optimization performance. In the manner just described, PEST could optimize 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 is always negative, which means it cannot be log-transformed. However if a new parameter is defined as the negative of the model-required parameter, PEST can optimize this new parameter, log-transforming it if necessary to enhance optimization 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 that fixed and tied parameters must also be supplied with a scale and offset, just like their adjustable (log-transformed and untransformed) counterparts.

- Before writing a parameter value to a model input file, PEST multiplies the value by the scale and then adds the offset.
- If you do not wish a parameter to be scaled and offset, enter its scale as 1 and its offset as zero.
- Fixed and tied parameters must also be supplied with a scale and offset, just like their adjustable counterparts.
- A parameter's scale and offset values are defined by the **SCALE** and **OFFSET** terms in the PEST control file (*projectname.pst*).

Parameter Change Limits

PEST cannot adjust a parameter 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 optimization iteration.

If the model under PEST's control exhibits reasonably linear behavior, 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 non-linear, the parameter upgrade vector may "overshoot" the objective function minimum, and the new value of the objective function may actually be worse than the old one. This is because 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 reduce 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 optimization iteration. Such limits may be defined as either relative-limited or factor-limited. However, log-transformed parameters must be factor-limited.

If a parameter is factor-limited, the maximum allowable change of the parameter value per iteration is defined as follows:

Let f represent the user-defined maximum allowable parameter factor change (f must be greater than one). Then if b_0 is the value of the parameter at the beginning of the optimization iteration, the value of the parameter at the beginning of the next optimization iteration, b, will lie between the limits

$$b_0/f \le b \le fb_0 \qquad (3.1a)$$

if b_0 is positive, and

$$fb_0 \le b \le b_0/f \qquad (3.1b)$$

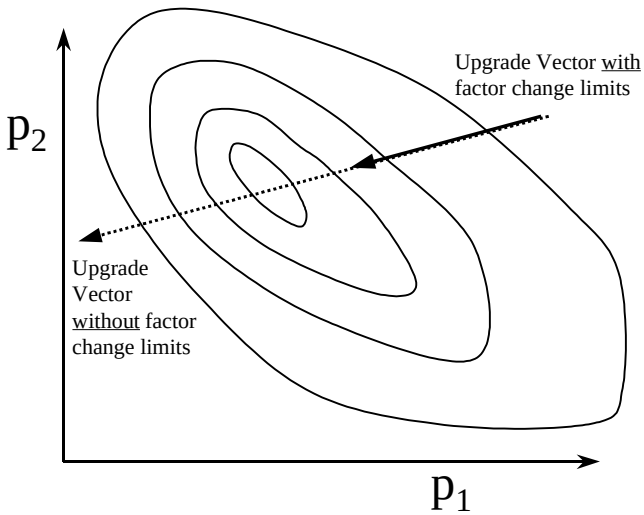


Figure 3.4: Two parameter example of how an upgrade vector without factor change limits can overshoot the minimum of the objective function.

if b_0 is negative.

The implication of equation (3.1) is that a parameter subject to factor-limited changes can never change sign.

On the other hand if the parameter change is relative-limited, the maximum allowable change of the parameter value per iteration is defined as follows:

Let r represent the user-defined maximum allowable relative parameter change for all relative-limited parameters. r can be any positive number. Then if b_0 is the value of a relative-limited parameter at the beginning of an optimization iteration, its value b at the beginning of the next optimization iteration will be such that

$$|b - b_0|/|b_0| \le r \qquad (3.2)$$

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

In implementing the conditions set by equations (3.1) and (3.2), PEST limits the magnitude of the parameter upgrade vector such that neither of these equations is violated. Naturally, if only one type of parameter change limit is featured in a current PEST run (i.e. parameters are all factor-limited or are all relative-limited) only the pertinent one of these equations is considered.

If, in the course of an optimization run, PEST assigns to a parameter a value, which is very small in comparison to its initial value, then either of equation (3.1) or (3.2) may place an undue restriction on subsequent parameter adjustments. Thus if b_0 for one parameter is very small, the changes to all parameters may be set intolerably small so that equation (3.1) or equation (3.2) is obeyed for this one parameter. To circumvent this problem, PEST provides an additional input variable, FACORIG, which allows the user to limit the effect that an unduly low parameter value can have in this regard. Thus, if the absolute value of a parameter is less than FACORIG times the parameter's initial absolute value and PEST wishes to adjust the parameter such that its absolute value will increase, then FACORIG times its initial value is substituted into equation (3.1) and the denominator of equation (3.2) for the parameter's current value b_0. A suitable value for FACORIG varies from case to case, but 0.001 is often appropriate. Note, however, that FACORIG is not used to adjust change limits for log-transformed parameters.

- PEST allows parameter changes to be either factor-limited or relative-limited.
- A **factor-limited parameter** is one whose **new value** is limited to a specified fraction of the value from the previous iteration.
- A **relative-limited parameter** is one whose **change** between iterations is limited to a specified fraction.
- Log-transformed parameters must be factor-limited.
- Factor-limited parameters can never change sign.
- For relative-limited parameters, if the specified fraction is greater than or equal to 1, the new value may become a minute fraction of the previous value (or even zero), without approaching the parameter change limit. For some models this may invalidate the assumption of model linearity.
- To control very small changes in parameter values, the parameter **FACORI** is used as a minimum fraction for a parameter change.
- A typical value for FACORIG is 0.001.
- FACORIG is not used to adjust change limits for log-transformed parameters
- The type of parameter change limit for each parameter is defined by **PARCHGLI** in the PEST control file (*projectname.pst*).

- The two input variables, **RELPARMAX** and **FACPARMAX**, provide the maximum allowed relative and factor changes limits for all relative-limited and factor-limited parameters, respectively.

Damping of Parameter Changes

Parameter over-adjustment and any resulting oscillatory behavior 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) and (2.24). 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 that are not log-transformed, define q_j for parameter j as

$$q_j=\beta u_j /(fb_j - b_j)$$

if u_j and b_j have the same sign, and

$$q_j=\beta u_j /(b_j - b_j /f)$$

if u_j and b_j have the opposite sign (3.3)

where b_j is the current value for the Goth 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

$$s_1 = p_i /p_{0i}$$

if $i = i_0$. Otherwise,

$$s_1 = 0 \qquad (3.4a)$$

$$s_2 = q_l /q_{0l}$$

if $l = l_0$. Otherwise,

$$s_2 = 0, \text{ and} \qquad (3.4b)$$

$$s_3 = \beta u_k / \beta_0 u_{0k}$$

if $k = k_0$. Otherwise,

$$s_3 = 0 \qquad (3.4c)$$

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

$$\rho = (3 + s)/(3 + |s|) \qquad (3.5a)$$

if $s \geq -1$. Otherwise,

$$\rho = 1/(2|s|) \qquad (3.5b)$$

Then, the oscillatory behavior of the parameter estimation process can be mitigated, by defining a new parameter upgrade vector **v** by

$$\mathbf{v} = \rho\beta\mathbf{u} \qquad (3.6)$$

Temporary Holding of Insensitive Parameters

The probability of a parameter estimation process running smoothly and efficiently decreases with the number of parameters being estimated. Part of the reason for this lies in the increased probability that several of the parameters are highly correlated. Under such circumstances the normal matrix may become singular, or almost singular, which means that the calculation of the parameter upgrade vector can become very imprecise.

In highly parameterized problems, the objective function is likely to be relatively insensitive to some parameters in comparison to other parameters. As a result, PEST may decide that large changes are required for certain parameters so that they can make a contribution to reducing the objective function. However, limits are set on parameter changes and these limits are enforced such that the magnitude (but not the direction) of the parameter upgrade vector is reduced, if necessary.

If a parameter is particularly insensitive, it may dominate the parameter upgrade vector, i.e. the magnitude of the change calculated by PEST for this parameter may be far greater than that calculated for any other parameter. However, when the change for this parameter has been reduced by its relative or factor change limits, other more sensitive parameters may not change much at all. The result is that at the end of the optimization iteration the objective function may have been hardly changed and subsequent convergence may be intolerably slow.

This phenomenon can be avoided by temporarily holding troublesome (i.e. insensitive) parameters at their current value for an iteration or two. Such parameters are then removed from the calculation of the parameter upgrade vector. Offending parameters can often be identified as those undergoing the maximum relative- or factor-limited changes during an optimization iteration. PEST records this information during a run and in WinPEST you can view the current sensitivity of all parameters during the run.

PEST records the "composite sensitivity" of each parameter to a parameter sensitivity file after every optimization iteration. The composite sensitivity is the magnitude of the column of the Jacobian matrix pertaining to that parameter modulated by the weight attached to each observation, or $\mathbf{S}_{ii}$ of equation (2.22). The parameters with the lowest sensitivities are the most likely to cause trouble.

In some cases, it may be necessary to hold several parameters in this way. For example, once a particular troublesome parameter has been identified and held, another insensitive parameter may in turn dominate the parameter upgrade vector. This can continue until the set of parameters has been reduced to a set of sensitive parameters. Now, once the objective function has been reduced, the held parameters can be released one at a time until the final optimized solution has been found.

After PEST calculates the Jacobian matrix, and immediately before calculating the parameter upgrade vector, PEST looks for a *projectnam* .HLD file. If it does not find it, PEST proceeds with its execution in the normal manner. However, if it finds this file, it reads it for the current optimisation iteration. Youcan edit the .HLD file at any time and PEST will read it at the next opportunity. Alternatively, the hold facility in WinPEST updates this file automatically

- The probability of a parameter estimation process running smoothly and efficiently decreases with the number of parameters being estimated.
- If a parameter is particularly insensitive, it may adversely dominate the parameter upgrade vector, making convergence intolerably slow.
- This problem can be avoided by temporarily holding insensitive parameters at their current value for an iteration or two.
- PEST looks for and reads the *projectname*.HLD file after it calculates the Jacobian matrix and immediately before it calculates the parameter upgrade vector.
- WinPEST provides an easy means of temporarily holding parameters during a PEST run.

Observation Groups

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 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 group is “drowned” by other information, or dominates the inversion process.

If prior information is used in the inversion process, PEST lists the contribution collectively made to the objective function by all prior information items. Again, this allows the user to assess the impact that prior information exerts on the objective function and hence on the inversion process.

- Each observation must be assigned to an observation group.
- PEST provides the contribution made by each observation group to the change in the objective function.
- Likewise, PEST provides the contribution made collectively by the prior information, if it is used.
- This information can be used to ensure that no observation group or prior information either drowns other groups, or dominates the inversion process.

Termination Criteria

PEST uses a number of different criteria to determine when to halt the iterative process. However, only one of them (when the objective function equals zero) guarantees that the objective function has indeed been minimized. In difficult circumstances, any of the other termination criteria could be satisfied even if the objective function is well above its minimum and the 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 as any to stop. If these criteria are properly set, you can be reasonably sure that when PEST terminates the parameter estimation process, either the optimal set of parameters 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 behavior of the objective function itself. If the objective function is changing very little, or not at all, over a number of successive iterations, the time has come to cease execution. PEST stops the process if the objective function has not changed by a minimum amount over a specified number of iterations. Alternatively, PEST stops the parameter iteration process if there has been no reduction in the objective function, below its current minimum value, for a specified number of “unsuccessful” iterations.

The second indicator of either convergence to the minimum of the objective function, or of the unlikelihood that further iterations will find a better minimum is the behavior of the adjustable parameters. If successive iterations are not significantly changing parameter values, there is probably little to gain in continuing with process. Therefore,

PEST will stop execution if the largest relative parameter change over a specified number of iterations has been less than a specified value.

Finally, PEST also requires an upper limit on the number of optimization iterations, which PEST will carry out.

Other termination criteria are set internally. PEST will terminate the optimization process if it calculates a parameter set for which the objective function is zero. PEST will also terminate, if the gradient of the objective function with respect to all parameters equals zero, 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. However, if PEST is currently calculating derivatives using forward differences and the option to use central differences is available, PEST will switch to central differences for greater derivatives accuracy before going on to the next iteration.

PEST terminates execution if:

- the objective function goes to zero.
- the gradient of the objective function with respect to all parameters equals zero.
- the parameter upgrade vector equals zero.
- all parameters are at their limits and the upgrade vector points out of bounds.
- the maximum number of iterations is reached (**NOPTMAX**).
- the objective function has not changed by a minimum amount (**PHIREDSTP**) over a specified number of iterations (**NPHISTP**).
- there has been no reduction in the objective function, below its current minimum value, for a specified number of “unsuccessful” iterations (**NPHINORED**).
- if the largest relative parameter change over a specified number of iterations (**NRELPAR**) has been less than a specified value (**RELPARSTP**)

The Calculation of Derivatives

The following section provides information on:

- The Forward and Central Differences,
- Parameter Increments for Calculating Derivatives and,
- How to Obtain Trustworthy Derivatives.

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.

Accuracy in derivative calculation is fundamental to PEST's success in optimizing parameters. Experience has shown that the most common cause of PEST's failure to find the global minimum of Φ in parameter space is the presence of round-off 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 derivative calculation pertain to parameter "groups". In PEST, each parameter must be assigned to such a parameter group. Assigning derivative variables to groups, rather than to individual parameters is simpler and requires less memory. In many instances, parameters naturally fall into one or more categories. For example, the hydraulic conductivity of each zone being estimated. However, if you wish to treat each conductivity zone differently, as far as the derivative calculation is concerned, this can be done by assigning each conductivity to its own group.

The simplest way to calculate derivatives is the method of forward differencing (see Figure 3.3). To calculate derivatives in this manner, first PEST varies each parameter in turn by adding an increment to its current value (unless the current parameter value is at its upper bound, in which case PEST subtracts the increment). Then PEST runs the model, reads the altered, model-generated observations and 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 optimization iteration, each optimization 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, as the minimum of the objective function is approached, often to reach this minimum PEST must calculate the parameters with a greater accuracy than that available 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 in this way are accurate enough for most occasions, so long as the parameter increments are chosen wisely. As three-point derivatives are normally calculated by adding an increment to the current parameter value and then subtracting

the same increment, the method is referred to as the method of "central" differences. If a parameter value is at its upper bound or lower bound, the parameter increment is subtracted or added, respectively, once and then twice, the model being run each time.

PEST uses one of three methods to calculate central derivatives (see Figure 3.3). In the first or "outside" method, only the two outer parameters are used to calculate the derivative of the objective function with respect to the current parameter value. This method yields a more accurate derivative value than the forward difference method because the (unused) current parameter value is at the center 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 The four alternative methods of derivative calculation in PEST 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 define a least-squares straight line of best fit through the three parameter-observation pairs and to calculate 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 central derivatives are used for all parameters, each optimization iteration requires that at least twice as many model runs be carried out as there are adjustable parameters. If the central method is used for some parameters and the forward method for others, the number of model runs will lie somewhere between the number of adjustable parameters and twice the number of adjustable parameters.

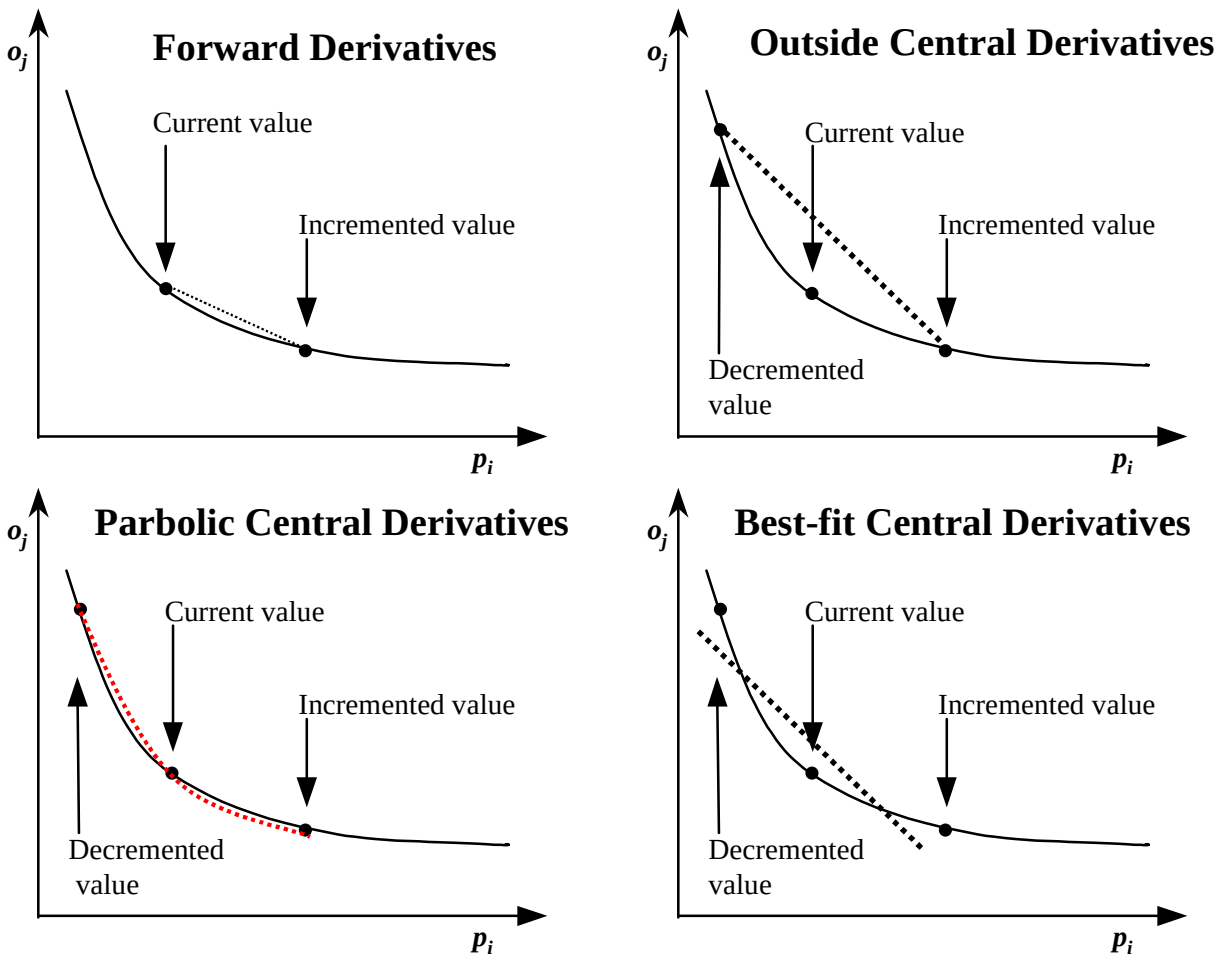


Figure 3.5: The four alternative methods of derivative calculation in PEST

- Round-off errors during the calculation of derivatives are the most common cause of PEST's failure to find the global minimum of the objective function.
- PEST variables that control the calculation of derivatives pertain only to parameter groups.
- PEST can calculate derivatives using forward differences or central differences, but using central differences requires twice as many model runs as forward differences.

Parameter Increments for Calculating Derivatives

PEST provides considerable flexibility in the way parameter increments are chosen, because of the importance of reliable derivative calculations. Mathematically, a parameter increment should be as small as possible so that the finite-difference method provides a good approximation to the theoretical derivative. However, if the increment is made too small, the accuracy of derivative calculations will suffer because of round off errors. For example, for forward differences, two, possibly large, numbers will be subtracted yielding 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 by which you can control how derivative increments are calculated, the increment type (INCTYP), the increment value or fraction (DERINC) and the minimum increment (DERINCLB). In PEST, there are three types of derivative increments: *absolute*, *relative* and *rel_to_max*. If the increment type is *absolute*, the user supplies the actual increment (DERINC) used for all parameters in the group. This increment is added to or subtracted from (for central derivatives) the current parameter value, when calculating derivatives with respect to that parameter. If the increment type is *relative*, the increment is calculated by multiplying the user-supplied increment value (DERINC) by the current absolute value of the parameter. Thus, the magnitude of the increment is adjusted as the parameter itself changes. If the increment type is *rel_to_max*, the parameter increment is calculated by multiplying the user-supplied value (DERINC) by the absolute value of the largest member of the parameter group. This can be a useful if the parameter values vary widely, including down to zero. The "relative" aspect of the *rel_to_max* type can lead to problems since the increment is calculated as a fraction of the maximum absolute value occurring within a group, rather than as a fraction of each parameter. Thus, an individual parameter can reach near-zero values without its increment simultaneously dropping to zero.

To protect against near-zero increments for *relative* and *rel_to_max* increments, PEST allows you to specify a minimum absolute increment (DERINCLB). This value is used in place of the calculated *relative* or *rel_to_max* increment if the calculated increment falls below the minimum increment value.

PEST also allows you to specify whether the derivatives are always calculated using the forward-difference method, (“always_2”) or by the central-difference method ("always_3"). Alternatively, if the derivative method is specified as “switch” then PEST will start the optimization using forward differences for all members of the group, and switch to central differences when the relative reduction in the objective function between optimization iterations is less than the specified tolerance (PHIREDSWH). This control over the method of calculating the derivatives is determined by the PEST group input variable FORCEN.

If the a derivative method is chosen that allows for central differences (“always_3” or “switch”) then two additional group variables are required. The first is the method used

to calculate the central derivative (DERMTHD), which can have the values "outside_pts", "parabolic" or "best_fit". The second variable is the increment multiplier for the three central derivative methods (DERINCMUL). Sometimes it is useful to employ larger increments for central derivative calculations than for forward derivatives calculations, especially where the model output versus parameter values is "bumpy" (see Figure 3.4). The parabolic method, which is a higher-order interpolation scheme, may allow you to place parameter values, and hence model-generated observation values, farther apart for calculating derivatives. This may increase the significance of the resulting differences from the derivative calculations. However, if the increment is raised too high, the precision of the derivatives must ultimately fall.

For increments calculated using the "relative" and "rel_to_max" methods, the minimum absolute increment (DERINCLB) has the same role in central derivatives calculation as it does in forward derivatives calculation. However, the minimum absolute increment is not multiplied by the increment multiplier (DERINCMUL).

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

PEST is also concerned that the derivative increment is not too large compared to the width of the parameter domain. To ensure this, PEST will object if the a parameter increment (either read directly as "absolute" or calculated from initial parameter values as "relative" or "rel_to_max") exceeds the parameter range (as defined by the parameter's upper and lower bounds) divided by 3.2. If during the estimation process the derivative increment exceeds the parameter range divided by 3.2, then PEST will automatically adjust the increment so that the parameter limits are not exceeded as the increment is added or subtracted from the current parameter value.

You must be careful when choosing an increment for a parameter to ensure that the parameter can be written to the model input file with sufficient precision to distinguish an incremented parameter value from one that has not been incremented. For example, if a parameter is written to a space in the template file that is four characters wide, and if the current parameter value is 0.01 and the increment is 0.0001, it will not be possible to discriminate between the parameter with and without its increment added. To rectify this situation, you must either increase the parameter field width in the template file (which would require you to change the template files) or increase the value of the increment.

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 template file. Also, for the purposes of derivative calculations, 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.

- A parameter increment should be as small as possible so that the finite-difference method provides a good approximation to the theoretical derivative.
- However, if the increment is made too small, the accuracy of derivative calculations will suffer because of round off errors.
- There are three types of derivative increments: *absolute*, *relative* and *rel_to_max* (**INCTYP**).
- *Absolute* – the user supplies the actual increment (**DERINC**) used for all parameters in the group
- *Relative* – the increment is calculated by multiplying the increment value (**DERINC**) by the current absolute value of the parameter.
- *rel_to_max* – the parameter increment is calculated by multiplying the user-supplied value (**DERINC**) by the absolute value of the largest member of the parameter group.
- PEST allows you to specify a minimum absolute increment (**DERINCLB**).
- You can specify whether the derivatives are always calculated using the forward-difference method, (“always_2”) or by the central-difference method ("always_3"), or by both (“switch”).
- For central difference derivatives you can specify the derivative method (**DERMTHD**), which can have the values "outside_pts", "parabolic" or "best_fit".
- If a parameter is log-transformed, it is wise that its increment be calculated using the "relative" method, though PEST does not insist on this.
- PEST will object if the parameter increment exceeds the parameter range divided by 3.2.
- You must be careful that the parameter can be written to the model-input file with sufficient precision to distinguish an incremented parameter value fro one that has not been incremented.

How to Obtain Derivatives You Can Trust

Precision in the calculation of the derivatives is essential for successful optimization. It is essential that any variables governing the numerical solution procedure be set in favor 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 derivative 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 parameter set corresponding to the global objective function minimum. Even if PEST is still able to find the global minimum (which it often will), it may require more optimization 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.

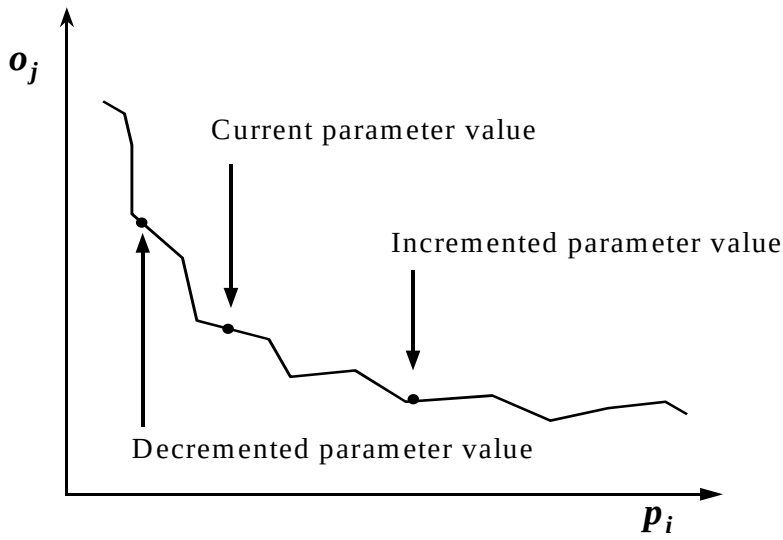


Figure 3.6: An example of model "granularity", where there is not a smooth (differentiable) function between the observations and the parameters.

For example, the matrix solvers used by MODFLOW (e.g. SIP, or PCG2) successively approximate the solution until "convergence" has been attained. The convergence is deemed acceptable when no element of the solution vector between successive iterations varies by more than the user-specified tolerance. lIf 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. However, as stated above, PEST may require more optimization iterations to find a solution, thereby removing any advantages gained in reducing the MODFLOW simulation time. Although PEST will happily attempt an optimization on the basis of limited-precision, model-generated observations, its ability to find an objective function minimum decreases as the precision of the 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 model output.

Unfortunately, model-generated observations may still be "granular" in that the relationship between these observations and the model parameters may be "bumpy" rather than continuous (see Figure 3.4). 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. Although a large increment incurs penalties due to the poor representation of the derivative by the finite

difference method (especially for highly non-linear models), using one of the central difference methods can mitigate this. 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.

PEST with MODFLOW and MT3D

Parameter Selection

Although non-linear parameter estimation is a powerful aid to model calibration, it will not work unless conditions are right. The following rules will help you decide whether PEST is likely to work or not in your particular case.

- Do not ask PEST to estimate more parameters than the observation dataset has the power to provide. A fundamental rule is that the number of adjustable parameters must not exceed the number of observations.
- Do not attempt a detailed parameterisation where borehole information is sparse. Even though the calibrated model may replicate borehole measurements well, the uncertainties associated with parameter estimates will be large and model predictions may be greatly in error.
- Avoid parameters that are highly correleated. This occurs when different combinations of parameter values result in almost the same model outcomes. Fortunately, ill-defined parameters or groups of parameters can be easily identified by their high uncertainty levels, large correlation coefficients and high eigenvalues (see Chapter 5). In general, the easiest way to avoid excessive parameter correlation is to keep the number of adjustable parameters to a minimum.
- Never try to estimate parameter combinations for which there is no unique solution. For example, in a steady-state model, if recharge is uniformly increased by a certain factor, model-generated heads will remain unchanged if transmissivity is increased by the same factor. Therefore, you should not attempt to simultaneously estimate transmissivity and recharge for a steady-state model using water levels as the only observations.
- Closely monitor the solution process if, in a transient model, you are attempting to simultaneously estimate two out of three of hydraulic conductivity, storage (or specific yield) and recharge. Although this is theoretically possible if heads and their variations with time are known everywhere, there may not be sufficient information to estimate two out of three of these parameter types because water level information is available only at discrete points and at discrete times.

- Similarly, be careful when trying to estimate multiple parameter types for MT3D, such as dispersivity and source concentration. Here the problem is exacerbated by the often high uncertainly associated with field measurements of solute concentration and the precision with which MT3D calculates concentrations.

In summary, the fewer parameter types and the less parameter values that you try to estimate, the better is PEST (or any other optimiser) likely to perform.

Modifying Model Input Files

PEST interfaces with a model through the models own ASCII input and output files. Each time PEST runs a model it first writes user-specified model input files using the parameter values which it wishes the model to use on that particular run. It knows where to write parameter values to input files through the use of model input file templates. For PEST to adjust a distributed parameter supplied to MODFLOW or MT3D through a two-dimensional array or cell-by-cell listing, a template must be constructed for the file which holds the array or listing. This is usually done by modifying a model input file, replacing parameter values with "parameter spaces" (comprising a parameter name enclosed by appropriate delimiters). Each parameter space denotes a contiguous set of characters on the model input file as belonging to a particular parameter. It also informs PEST of the number of digits which it may use to write the number representing the parameter.

Table 1: Template example for a two-dimensional array comprised of four different numbers

1.2345 1.2345 1.2345 1.2345 1.2345 6.7543 6.7543 6.7543 1.2345 1.2345 1.2345 1.2345 1.2345 6.7543 6.7543 6.7543 1.2345 1.2345 1.2345 9.6521 9.6521 6.7543 6.7543 6.7543 1.2345 1.2345 1.2345 9.6521 9.6521 9.6521 6.7543 6.7543 1.2345 1.2345 1.2345 9.6521 9.6521 9.6521 9.6521 6.7543 8.4352 1.2345 1.2345 9.6521 9.6521 9.6521 9.6521 9.6521 8.4352 8.4352 1.2345 9.6521 9.6521 9.6521 9.6521 9.6521 8.4352 8.4352 8.4352 9.6521 9.6521 9.6521 9.6521 9.6521 8.4352 8.4352 8.4352 8.4352 9.6521 9.6521 9.6521 9.6521
par1 # # par1 # # par1 # # par1 # # par1 # # par2 # # par2 # # par2 # # par1 # # par1 # # par1 # # par1 # # par1 # # par2 # # par2 # # par2 # # par1 # # par1 # # par1 # # par3 # # par3 # # par2 # # par2 # # par2 # # par1 # # par1 # # par1 # # par3 # # par3 # # par3 # # par2 # # par2 # # par1 # # par1 # # par1 # # par3 # # par3 # # par3 # # par3 # # par2 # # par4 # # par1 # # par1 # # par3 # # par3 # # par3 # # par3 # # par3 # # par4 # # par4 # # par1 # # par3 # # par3 # # par3 # # par3 # # par3 # # par4 # # par4 # # par4 # # par3 # # par3 # # par3 # # par3 # # par3 # # par4 # # par4 # # par4 # # par4 # # par3 # # par3 # # par3 # # par3

For a spatially distributed parameter occupying a two-dimensional array the model domain must be subdivided into a handful of zones where the parameter is constant. If each number in the array is replaced by an appropriate parameter space, the array of numbers as represented in the model input file becomes an array of parameter spaces. Each zone of parameter constancy within the array is then identified as having the same parameter name.

The first part of Table 1 illustrates a two-dimensional array of numbers subdivided into four zones of equal value. The second part of T able1 shows part of a template file constructed from it. Before PEST runs the model, it replaces the parameter spaces found in the template file by the current values pertaining to those parameters, thus building an array consisting of four separate numbers and defining four separate zones of parameter constancy

For parameters supplied to MODFLOW or MT3D on a cell-by-cell basis the cells can be divided into zones of similar value in the same way. For example, Table 2 shows part of a MODFLOW .DRN file for the Drain Package.

Table 2: Template example for part of the input to MODFLOW's DRN package.

```
 1     19     43 2.000E+01 3.000E+00
 1     20     43 2.000E+01 3.000E+00
 1     21     43 2.000E+01 3.000E+00
 1     22     44 2.000E+01 3.000E+00
 1     23     45 2.000E+01 3.000E+00
 1     24     46 2.000E+01 5.000E+00
 1     25     46 2.000E+01 5.000E+00
 1     26     46 2.000E+01 5.000E+00
 1     27     46 2.000E+01 5.000E+00
 1     28     45 2.000E+01 5.000E+00
 1     29     44 2.000E+01 5.000E+00
 1     30     43 2.000E+01 5.000E+00
 1     31     43 2.000E+01 5.000E+00
```

```
 1     19     43 2.000E+01 # con1  #
 1     20     43 2.000E+01 # con1  #
 1     21     43 2.000E+01 # con1  #
 1     22     44 2.000E+01 # con1  #
 1     23     45 2.000E+01 # con1  #
 1     24     46 2.000E+01 # con2  #
 1     25     46 2.000E+01 # con2  #
 1     26     46 2.000E+01 # con2  #
 1     27     46 2.000E+01 # con2  #
 1     28     45 2.000E+01 # con2  #
 1     29     44 2.000E+01 # con2  #
 1     30     43 2.000E+01 # con2  #
 1     31     43 2.000E+01 # con2  #
```

The drain has been subdivided into two zones in each of which the conductance is assumed uniform. (Note that in this example, the parameterization would probably benefit by tying all of the conductances to one conductance.

More detailed description of the syntax and structure of the PEST template files can be found in “PEST Template Files” in Appendix B.

Visual MODFLOW’s Template Files

Visual MODFLOW takes care of creating template files for the parameters that you select in the PEST Control dialogue. In this dialogue, you can currently select spatially variable anisotropic conductivities, storage parameters and recharge. The parameters that you select here are Visual MODFLOW parameters - not MODFLOW parameters. This means that you can select vertical hydraulic conductivity whereas in MODFLO this term is lumped into the vertical conductance variable.

Visual MODFLOW builds the MODFLOW input files before each run by using a combination of PERL source files (.SRC files) and template files that are written in C. PEST substitutes the current parameter value into the template file, which then creates the MODFLOW input file in the format outlined by the .SRC files.

An example of a Visual MODFLOW template file can be found in Appendix B.

Reading Output Files

PEST Instruction Files

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 binary files.

Unfortunately, observations cannot be read from model output files using the template concept, since neither MODFLOW nor MT3D cannot be relied upon to produce an output file of identical structure during each model run. So instead of using an output file template, you must provide PEST with a list of instructions on how to find observations in the output files (see Ta ble3).

Basically, PEST finds observations in a model output file in the same way that you would. You run your eye down the file looking for something you recognise - a "marker". If this marker is properly selected, observations can usually be linked to it. For example, if you are looking for the output after 100 days, you may look for

TIME = 100 DAYS

A particular outcome for which you have a corresponding field measurement may then be found, for example, between character positions 23 and 30 on the 4th line following the marker. For output files, a marker may be unnecessary as the default initial marker is the top of the file.

Table 3: Example output file and corresponding PEST instruction file.

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

```
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
```

During translation Visual MODFLOW creates the instruction files for reading the MODFLOW heads (*projectname*.inh), MT3D concentrations (*projectname*.inc) and ZoneBudget flows (*projectname*.inz).

For more detail on the format, structure and syntax of instruction files, see “How PEST Reads Model Output Files” in Appendix B.

Interpolating Model Outcomes to Borehole Locations

The data available for groundwater model calibration usually consists of water level or solute concentration measurements made at boreholes scattered throughout the model domain. The borehole observations may be at a single time, or may have been taken over a period of time. In either case, model calibration requires the adjustment of model parameters until the water levels or concentrations generated by the model at these borehole locations correspond as closely as possible with those actually observed.

It is essential for good PEST performance that model-calculated values correspond to the field observations as close as possible. Unfortunately neither MODFLOW nor MT3D interpolates its calculated heads or concentrations to the actual borehole locations and measurement times. Therefore, PEST includes the two programs, MODBORE for MODFLOW and MT3BORE for MT3D, to do the necessary spatial interpolation. MODBORE and MT3BORE obtain the heads, drawdowns and concentrations calculated by MODFLOW and MT3D by reading the unformatted head, drawdown or concentration files produced by these models.

Thus, the "model" run by PEST actually consists of at least two programs. For MODFLOW calibration PEST runs MODFLOW followed by MODBORE. For MT3D calibration PEST runs MT3D followed by MT3BORE. For joint MODFLOW/MT3D calibration PEST runs MODFLOW followed by MODBORE followed by MT3D followed by MT3BORE.

MODFLOW and MT3D Output Timing

Both MODBORE and MT3BORE interpolate all arrays found in the MODFLOW or MT3D unformatted output files to the borehole locations specified in Visual MODFLOW. For steady-state MODFLOW simulations there is only one head and drawdown array created. However, for transient simulations, Visual MODFLOW also interpolates the output from MODBORE and MT3BORE to the observed times before PEST uses the data.

Visual MODFLOW allows you to specify the times at which you want MODFLOW and MT3D output arrays printed. This is very useful for PEST. If the output times correspond to the measurement times, then errors in temporal interpolation can be minimized and the accuracy of the PEST-calculated derivatives improved.

MODBORE and MT3BORE Spatial Interpolation

MODFLOW and MT3D calculate the heads and concentrations at the centroid of each cell. MODBORE and MT3BORE use a bilinear interpolation scheme to interpolate these calculated nodal values to the actual borehole locations.

For each borehole location, the four neighbouring cell centroids in the the layer are determined. Then, the heads, drawdowns or concentrations at these nodes are interpolated to the borehole location. If the four surrounding nodes have row and column numbers (i,j), (i+1,j), (i,j+1) and (i+1,j+1), the head at the borehole location is calculated as:

$$h = \frac{x_2 \cdot y_2 \cdot h_{i,j} + x_1 \cdot y_2 \cdot h_{i,j+1} + x_2 \cdot y_1 \cdot h_{i+1,j} + x_1 \cdot y_1 \cdot h_{i+1,j+1}}{XY}$$

where:

$h_{i,j}$ is the head at the centre of cell (i,j),

$x_1 = x_p - x_{i,j}$

$x_2 = x_{i,j+1} - x_p$

$y_1 = y_{i,j} - y_p$

$y_2 = y_p - y_{i+1,j}$

$X = x_{i,j+1} - x_{i,j}$

$Y = y_{i,j} - y_{i+1,j}$

(x_p,y_p) are the x and y coordinates of the measurement point, and

$(x_{i,j},y_{i,j})$ are the x and y coordinates of the centre of cell (i,j).

In the above expression all coordinates are expressed in a Cartesian system where the x direction corresponds to the positive row direction (i.e. the direction of increasing column index) and the y direction corresponds to the negative column direction (i.e. the direction of decreasing row index). Note that no interpolation takes place in the vertical (i.e. inter-layer) direction.

The above interpolation scheme is slightly modified if a borehole is close to the edge of the model grid, near an inactive zone, or near a dry cell, such that the cell does not have four neighbours. If a borehole lies within an inactive or dry cell, or is outside the grid altogether, a comment is written to the MODBORE or MT3BORE output file.

MODBORE uses the following files:

modbore.in	Input file containing file names etc. read by MODBORE
modbore.in.src	Standard file from which modbore.in is created each time
projectname.spc	Input file containing the MODFLOW grid definition
projectname.lst.flo	List of head observation wells
projectname.crd.flo	Coordinate file for head observation wells
projectname.hds	Binary heads file from MODFLOW
modbore.out	Output file containing spatially interpolated heads
projectname.hob	Output file with temporally interpolated heads from modbore.out

MT3BORE uses the following files

mt3bore.in	Input file containing file names etc. read by MT3BORE
mt3bore.in.src	Standard file from which mt3bore.in is created each time
projectname.spc	Input file containing the MODFLOW grid definition
projectname.lst.cnc	List of concentration observation wells
projectname.crd.cnc	Coordinate file for concentration observation wells
projectname.ucn	Binary concentration file from MT3D
mt3bore.out	Output file containing spatially interpolated concentrations
projectname.cob	Output file with temporally interpolated concentrations from mt3bore.out

MODBORE and MT3BORE as an Aid to Contouring

MODBORE and MT3BORE read the MODFLOW or MT3D unformatted output files, interpolate the heads and concentration arrays to the borehole locations and write the interpolated values to the modbore.out and mt3bore.out files, respectively. In addition to the interpolated values, these files also contain the borehole coordinates. This means that the data in these files can be read by any contouring program such as SURFER by Golden Software. You may have to remove the header to make it work with your contouring package or extract the timestep of interest if you have a transient simulation. Such a contour map can be very useful in model calibration when it is compared to field measurements contoured with the same package.

Using MODBORE and MT3BORE with PEST

When calibrating MODFLOW, it is the temporally interpolated MODBORE output which PEST must use. Similarly, if you are calibrating MT3D, PEST must use the temporally interpolated output from MT3BORE. Visual MODFLOW automatically runs MODBORE and MT3BORE if needed and creates the instruction files (see “PEST Instruction Files” on p age45) for these output files.

To use MODBORE or MT3BORE outside of Visual MODFLOW, please see the PEST documentation included as a .pdf file on your installation CD.

4

4 - PEST in Visual MODFLOW

The parameter estimization process is now an integral part of the Visual MODFLOW environment. The Chapter deals with

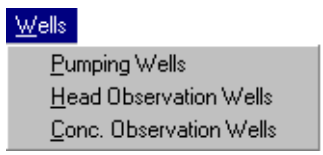
- Assigning Observations to Model Outputs,
- Choosing the Parameters to Optimize,
- Assigning Prior Information,
- Assigning the Objective Function,
- Controlling the PEST Run and,
- Starting the PEST Run.

Assigning Observations to Model Outputs

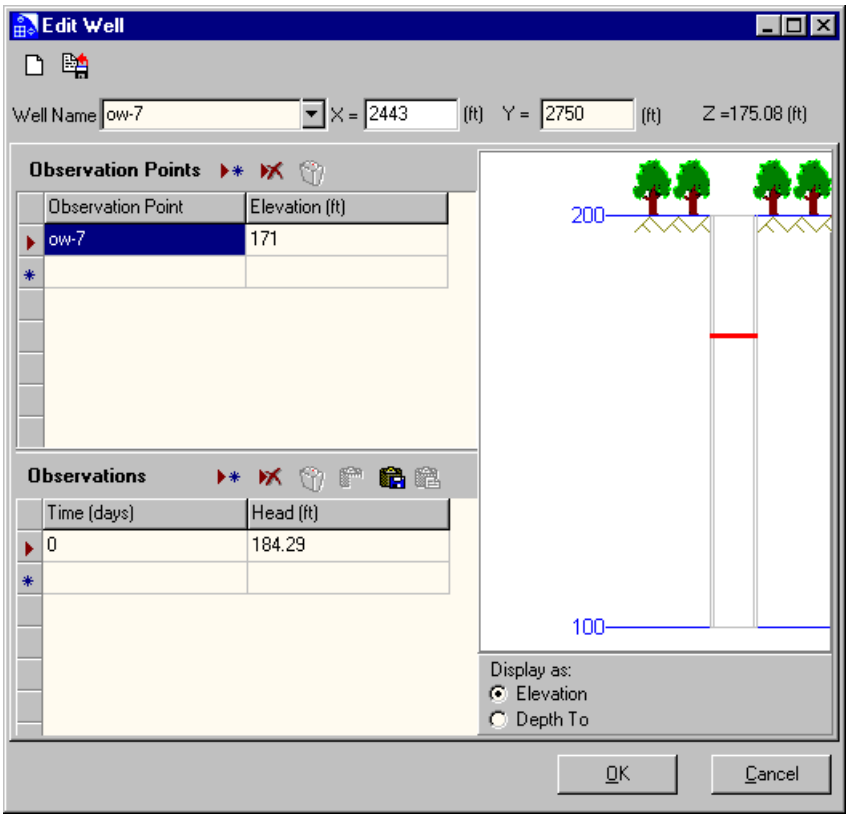
Visual MODFLOW allows you to relate observations of head, concentration and flow to model output values. All observations are input to Visual MODFLOW in the Input module.

Head and Concentration observations

Head and concentration observations are input in the Wells mode as seen below



When either head or concentration observations are selected from the Wells drop-down menu, the observation values can be input in the following dialogue. This dialogue can be accessed whenever observations are being added or edited.



In the Borehole Edit dialogue, you must provide a well name and the map x, y coordinates of the well. If you add a new well, it will not yet have an observation point. The observation point can be added by clicking in the “Observations Points” section and by default a point will appear at the mid-point of the diagram on the right hand side. The elevation of the observation point is important as this defines the model cell that corresponds to the observed value. Typically, the observation point is defined at the middle of the screened interval in a well, but this may differ depending on the stratigraphy and the manner in which the well screen is installed. For example, if the well is screened over the entire depth but the top 10 metres of the profile is fine silt and clay, then the observation point may be defined at the midpoint of the more permeable strata. The observation point that is defined is independent of the model grid. If you refine the grid or move the model layers, then Visual MODFLOW will automatically assign your observations to the appropriate model cell based on where you define the observation point.

Visual MODFLOW allows you to have multiple observation points in a single borehole. You add observation points by clicking on the Add icon and then either typing in the elevation or clicking on the borehole and moving the red bar to the

appropriate level. For each observation point you can type in or import a list of observed values.

Flow Observations

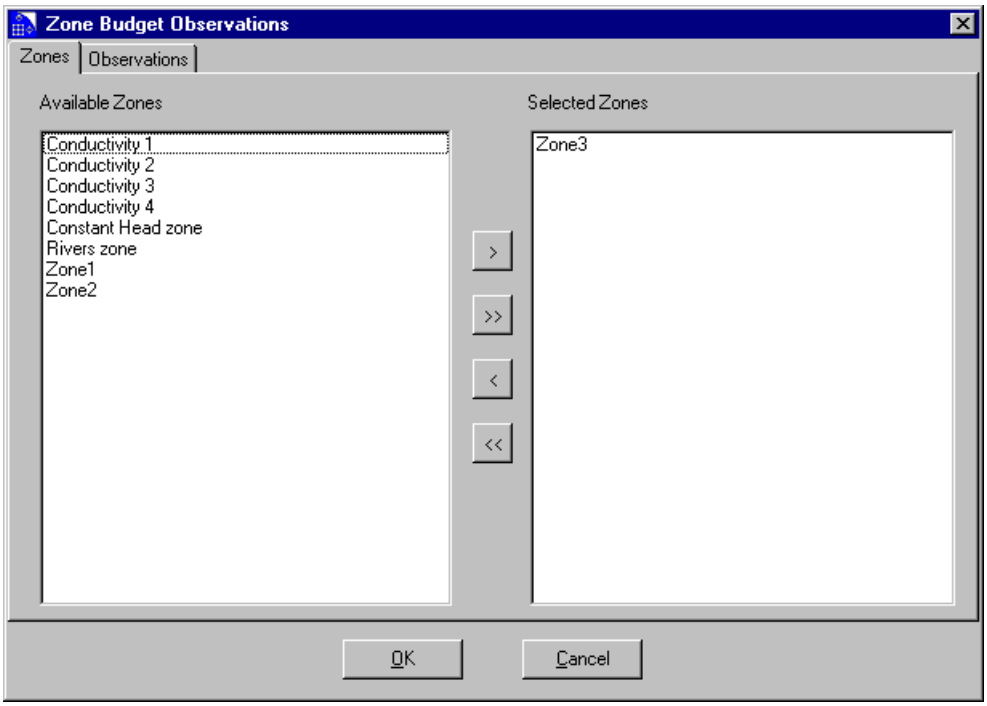
For flow observations, such as baseflow to a stream, the observations must be input in the Zone Budget mode which is accessed from the **ZBud** item on the top menu bar.

Zone Budget is a program developed by the USGS, which calculates the zone-to-zone flows from the MODFLOW output files. For example, if you divide a river into a number of Zone Budget zones, Zone Budget will calculate the amount of water entering each river zone from the model. In this manner, you will be able to map out gaining and losing stretches of the river and calibrate your model to stream baseflow measurements from the field.

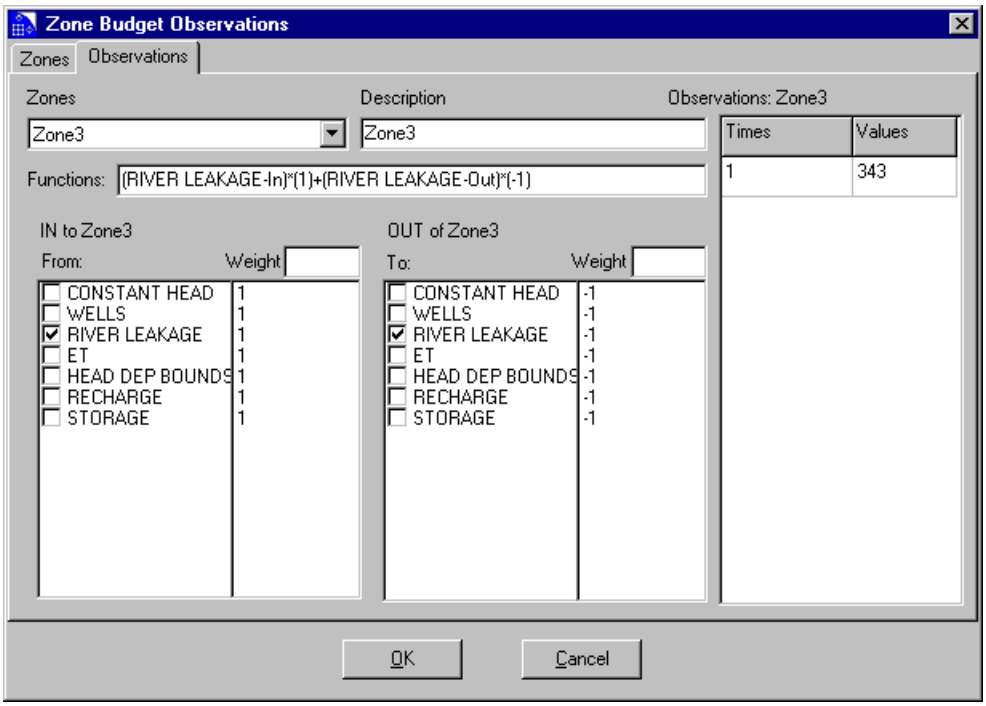
In Visual MODFLOW, Zone Budget zones are added to a model in the same manner as all other zones, such as conductivity zones and recharge zones. The only important difference is that the model does not need to be re-run if a new zone is added or if the zone boundaries are modified. Only the Zone Budget program needs to be run, since Zone Budget calculates all of its flows from the MODFLOW output files.

Zone Budget automatically calculates the external flows between the model domain and the boundary conditions defined in the model. In addition to these basic zones, Visual MODFLOW also creates separate zones for each of the defined conductivity zones and each of the boundary conditions. Thus, Zone Budget automatically calculates the sum amount of water entering and leaving the model through the river nodes. However, Visual MODFLOW also creates a proper Zone Budget zone for the cells that are defined as river cells. Thus, the amount of water entering and leaving the river cells can also be calculated, since not all water that enters a river cell leaves the model.

Before the user is allowed to enter flow observations in Zone Budget, the MODFLOW filees must be translated. Flow observations that correspond to Zone Budget flows are added by clicking on the Observation button on the side menu bar, which brings up the following dialogue:



In the Zones tab of the Zone Budget Observation dialogue all of the available zones are listed. Since the number of zones could be more than a hundred in a complex model, the Zones tab allows you to select which zones to use in the Zone Budget calculations. In the Observations tab, the upper combo box contains the list of selected zones from the Zones tab, as does the Zones List in the Output module.



Typically, you will have a baseflow measurement to a stream, which should equal the amount of flow entering or leaving the model through a particular stream section. To allow PEST to compare this baseflow measurement to the output from Zone Budget, you must:

[1] Create a new, user-defined zone along the stream section (e.g. *Zone 4*). This is essential if there are several conductivity zones along the stream.

[2] Select *Zone 4* and the *Rivers Zone* from the Available Zones list on the Zones tab.

[3] In the Observation tab, select *Zone 4* from the Zones combo box

[4] Select the checkbox beside *River Leakage* in the list under "OUT". Ensure that the weight is +1 if your measurements are positive values.

[5] Input the time from the beginning of the simulation if your model is transient (or any time if it is steady-state) and your measured baseflow rate.

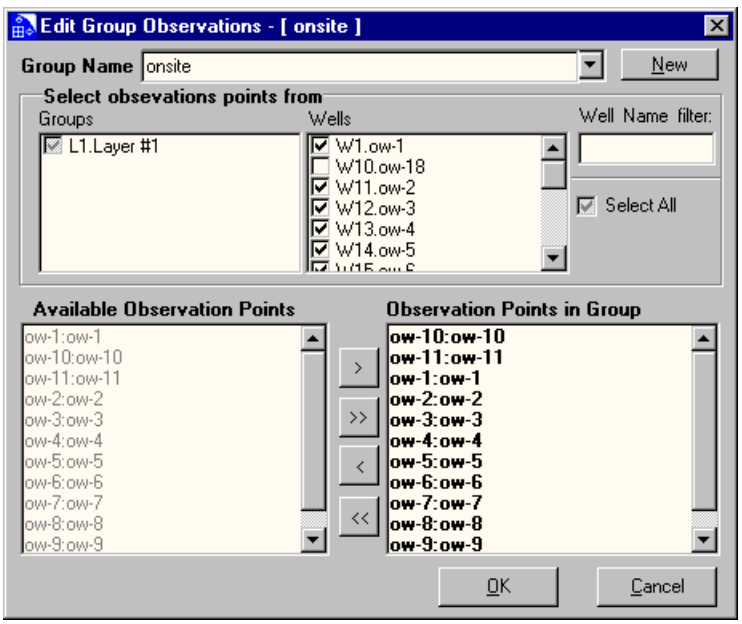
Occasionally, the way Zone Budget calculates flows may not correspond to the way the flows were measured in the field. For this case, the Observation tab contains a means of adding and subtracting the Zone Budget output such that the sum corresponds to the way the data was measured. Thus, for each zone that you select from one of the lists, both the zone and its weight are added to the function line. PEST adds the Zone Budget

output values according to the defined function prior to comparing the Zone Budget output to the observations.

Observation Groups

The site observations can be naturally divided into head, concentration and flow observations. However, in Visual MODFLOW each type of observation may be grouped within these broader classifications. There is two reasons for this. First of all, you can use the group function to more easily assess your model calibration in specific parts of you model, for example in each aquifer or inside your site and outside your site. In addition to this, PEST allows you to calculate the contribution of each group to the overall objective function. In this case, the goal is to modify the weights assigned to each group such that one group of observations does not dominate the calculation of the objective function. For example, you may have a lot of wells on your site, but relatively few everywhere else. In this case, you may want to balance the weights so that the observations outside of your site contribute more fairly to the overall calibration.

The observation groups are assigned and edited in the **Input** mode by clicking on the **[Edit Groups]** button on the left hand tool bar when you have selected **Heads** or **Concentration** observations. This will activate the following **Edit Group Observations** dialogue.



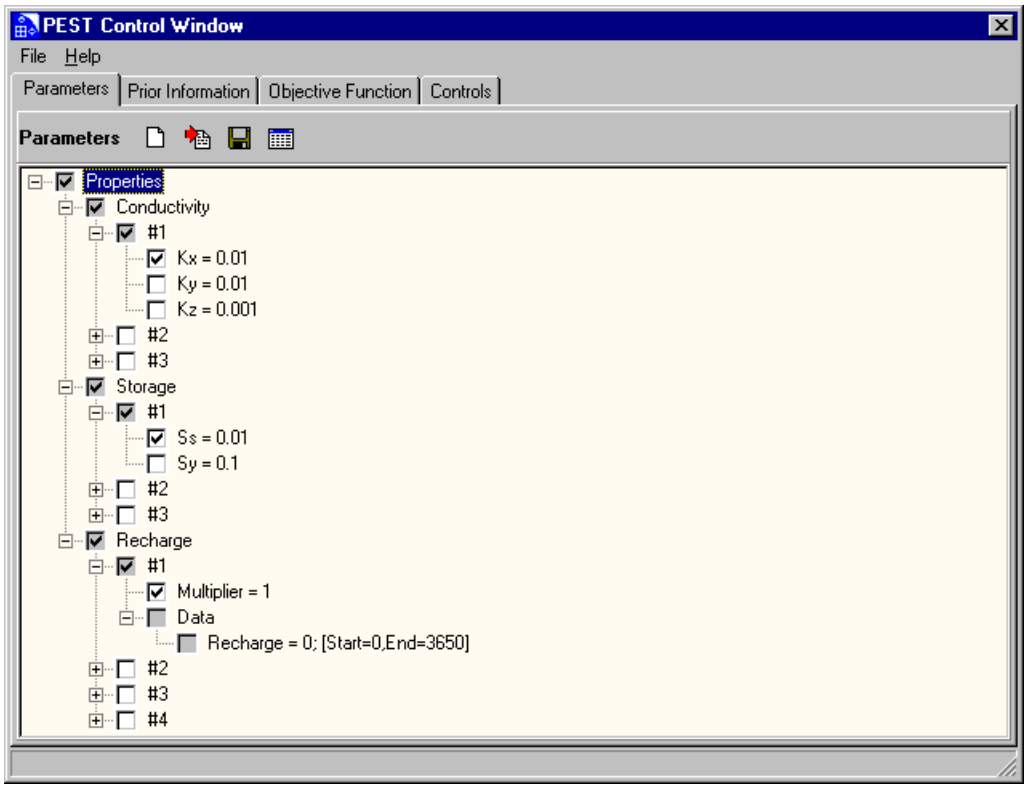
In this dialogue, you can select an existing user group to edit or add a new user group. Under **Groups** is listed all of the available observation groups for flow or transport,

including user-defined groups. If you select a group in this list then all of the wells that belong to this group will be selected in the **Wells** list. The Wells list includes all of the observation wells in the model. When a well in this list is selected the observation points associated with the well are listed in the **Available Observation Points** list, which means that all observation points for a multiple-level piezometer will be listed. From the list of **Available Observation Points**, you can move individual observation points to the list of observation points that are in the current group.

Choosing the Parameters to Optimize

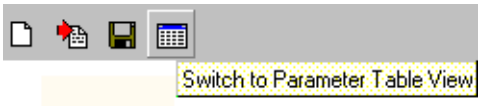
Once the model is working and your field observations have been specified in the Input module, then you are ready to run PEST. From the Run module, you can select PEST from the top menu bar after selecting Run in the Main Menu.

This will bring up the PEST Control Window.

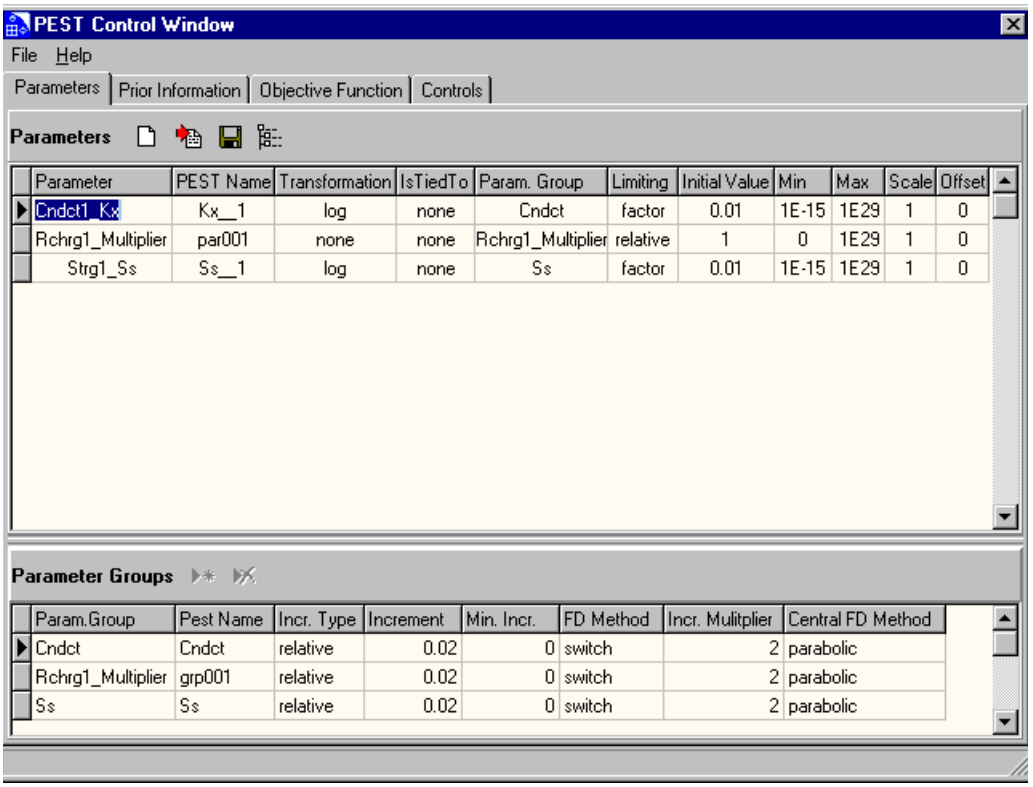


By default the first view in the PEST Control dialogue is the tree view of the parameters that are available for PEST to estimate. From this tree view you can select your parameters by clicking on the check boxes beside each parameter. In the current version of Visual MODFLOW only Conductivity, Storage and Recharge are available in WinPEST. This list of available parameters will be extended in future versions to include virtually all Visual MODFLOW flow and transport parameters. Similarly, the current version of WinPEST is only capable of estimating MODFLOW parameters, but future versions will include parameters from all models supported by Visual MODFLOW, including MODPATH, MT3Dxx and RT3D.

After you have selected your parameters, you can switch to the table view by clicking on the Table icon in the menu bar:



this will change the display to the following:



In this view, you can modify any of the PEST variables for each of the parameters you have selected. The Parameters table contains all the parameters that you selected in the tree view and default values for all of the PEST variables. Summary outlines for each of the Visual MODFLOW parameters are presented below. For a more detailed description refer to the indicated chapter and page numbers.

Parameters

Parameter

The long name assigned by Visual MODFLOW to the parameters selected in the tree view.

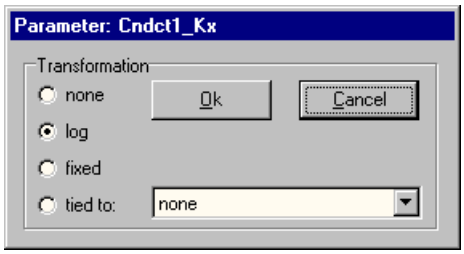
PEST Name - PARNME

Parameter label used in the PEST input and output files.

Transformation - PARTRANS and IsTiedTo

The Transformation column defines whether the parameter is logarithmically transformed (log) or not (none), tied to a parent parameter (tied) or fixed at its initial value (fixed).

If you select the Transformation or IsTiedTo fields you can click on the small button that appears and the following dialogue will appear:



If you select **tied to:** then you can select the parent parameter from the combo box. The name of the parent parameter will appear in the **IsTiedT** column. If you subsequently untie, fix or transform the parameter, the name in the combo box and in the IsTiedTo column will not change. This is to allow you to tie and untie parameters during the optimisation process without having to select the parent parameter each time. Since, PEST does not allow a parameter to be tied to a fixed or other tied parameter, only the available parameters are listed in the combo box.

In many cases, the linearity assumption, on which PEST is based, is more valid when certain parameters are log-transformed. This means that the log-transformation of some parameters can often make the difference between success and failure of the estimation process. However, a parameter that can become zero or negative during the estimation

process must not be log-transformed. This can be corrected using an appropriate **Scale** and **Offset**.

If a parameter is tied to a parent parameter, the parameter “piggy-backs” on the parent parameter during the estimation process. That is the ratio between the initial values of the parameter and its parent remain constant throughout the estimation process.

- If the parameter is log-transformed then the **Initial value**, **Min.**, **Max.**, **Scale** and **Offset** values must be untransformed. That is, they must **not** be log-transformed.
- If the parameter is log-transformed then the covariance, correlation coefficients and eigenvector values refer to the log of the parameter. However, the parameter estimates and confidence intervals refer to the untransformed parameter.
- If you to fix a parameter, its value will be fixed at its initial value and it will not be part of the estimation process.

For more detailed information on **Parameter Transformation** and **Fixed and Tied Parameters** see pag e23 and page24 of the PEST User’s Manual.

Param. Group - PARGP

This is the parameter group to which the parameter belongs. The available parameter groups are listed in the combo box and below the Parameter Groups section in the dialogue. Additions to the list of available parameter groups must be made in the Parameter Groups section of the dialogue.

In PEST, input variables affecting derivative calculations pertain to parameter groups. Each parameter must be assigned to such a parameter group Assigning derivative variables to groups, rather that to individual parameters is simpler and requires less memory.

Limiting - PARCHGLIM

PEST allows parameter changes to be either factor-limited (factor) or relative-limited (relative). A **factor**-limited parameter is one whose **new** value is limited to a specified

fraction of the value from the previous iteration. A **relative**-limited parameter is one whose **change** between iterations is limited to a specified fraction.

- Log-transformed parameters must be factor-limited.
- Factor-limited parameters can never change sign.
- For relative-limited parameters, if the specified fraction is greater than or equal to 1, the new value may become a minute fraction of the previous value (or even zero), without approaching the parameter change limit. For some models this may invalidate model linearity assumptions.
- The PEST control parameters FACORIG, PARCHGLIM, RELPARMAX, and FACPARMAX can be modified in the Controls Tab.

For more detailed information on the **Parameter Change Limits** see page 27 of the PEST User's Manual.

Initial Value - PARVAL1

This is the initial value for the parameter estimation process and is equal to value assigned in the Input module. If the parameter is fixed then the value for this parameter remains constant at its initial value during the estimation process.

Min and **Max - PARLBND** and **PARUBND**

The Min and Max are the lower and upper bounds of the parameters respectively.

- The lower and upper bounds should be chosen wisely.
- The default values are 1e-15 and 1e29 respectively.
- The upper and lower bounds are ignored for fixed and tied parameters.
- If an updated parameter value is outside of its bounds, PEST temporarily holds the parameter at its boundary value.
- The strategy that PEST uses, allows PEST to search along the bounds of the parameter domain looking for the minimum value of the objective function.

For more detailed information on **Upper and Lower Parameter Bounds** see pag e25 of the PEST User's Manual.

Scale and **Offset - SCALE** and **OFFSET**

The scale and offset can be used to modify the numerical value of a parameter to make it more amenable to parameter estimation.

- Just before writing a parameter value to a model input file, PEST multiplies the value by the scale and then adds the offset.
- If you do not wish a parameter to be scaled and offset, enter its scale as 1 and its offset as zero.
- Fixed and tied parameters must also be supplied with a scale and offset, just like their adjustable counterparts.

For more detailed information on **Scale and Offset** see page 26 of the PEST User's Manual.

Parameter Groups

The Parameter Groups table lists the different groups that PEST will assign the Visual MODFLOW parameters to. The Parameter Groups are used for grouping parameters whose derivatives share common characteristics. The derivatives for all the parameters in a parameter group will be calculated using the same method. For more information on the derivative calculations see the **The Calculation of Derivatives** section in the PEST User's Manual.

Param. Group

This is the long name for the group.

PEST Name - PARGPNME

This is the group name that appears in the PEST input and output files.

Incr. Type - INCTYP

There are three types of derivative increments: *absolute*, *relative* and *rel_to_max*.

Absolute – the user supplies the actual increment (DERINC) used for all parameters in the group

Relative – the increment is calculated by multiplying the increment value (DERINC) by the current absolute value of the parameter.

rel_to_max – the parameter increment is calculated by multiplying the user-supplied value (DERINC) by the absolute value of the largest member of the parameter group.

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

Increment - DERINC

This is the value of DERINC defined above. A parameter increment should be as small as possible so that the finite-difference method provides a good approximation to the theoretical derivative. However, if the increment is made too small, the accuracy of derivative calculations will suffer because of round off errors.

- PEST will object if the parameter increment exceeds the parameter range divided by 3.2.
- a suitable value for DERINC is often 0.01 if INCTYP is *relative* or *rel_to_max*.

Min. Incr. - DERINCLB

To protect against near-zero increments for *relative* and *rel_to_max* increments, PEST allows you to specify a minimum absolute increment. This value is used in place of the calculated *relative* or *rel_to_max* increment if the calculated increment falls below the minimum increment value.

- If you do not want to have a minimum increment, use zero for DERINCLB.
- If INCTYP is “absolute”, DERINCLB is ignored.

FD Method - FORCEN

In this column, you can specify whether the derivatives are calculated using the forward-difference method, (“always_2”) or by the central-difference method ("always_3"), or by both (“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. In this case, to fill the columns of the Jacobian matrix corresponding to members of the group, as many model runs as there are adjustable parameters in the group will be required. If FORCEN is "always_3", it will require twice as many model runs to fill the same columns in the Jacobian matrix. 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", PEST will calculate the derivatives beginning with the forward-difference method and switch to the central method when the change in the objective function becomes small enough. For all switchable parameter groups, PEST

will switch to the central-difference method on the iteration after the relative change in objective function becomes less than PHIREDSWH. PHIREDSWH is defined in the Controls Tab of this dialogue.

In most instances the most appropriate value for FORCEN is "switch". This allow speed to take precedence over accuracy in the early stages of the optimisation process when accuracy is not critical. Accuracy takes precedence over speed later in the process when the derivatives need to be calculated with as much accuracy as possible. This is especially true when parameters are highly correlated and the normal matrix thus approaches singularity.

Incr. Multiplier - DERINCMUL

If derivatives are calculated using one of the three-point central-difference 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, you may want to increase the value of the increment for this process over that used for forward-difference derivative calculation. The real variable DERINCMUL allows you to do this. If a three-point derivative method is used, the value of DERINC is multiplied by DERINCMUL. This happens 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 the "absolute" increment type.

For many models, the relationship between observations and parameters is in theory continuously differentiable. However, in reality it is often "bumpy" (see Figure 9.4). In such cases, the use of parameter increments which are too small may lead to highly inaccurate derivatives, 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 the finite-difference method can approximate the derivatives. By definition, the derivative is 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 when you otherwise can not.

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_2", "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 a three-point method is used, you should set DERINCMUL to a value less than 1.0. Normally, a value between 1.0 and 2.0 is satisfactory.

Central FD Method - DERMTHD

If you are using central finite-differences to calculate the derivatives you can specify the three-point derivative method that is used: "outside_pts", "parabolic" or "best_fit". See Figure 9.3 for a comparison of these methods.

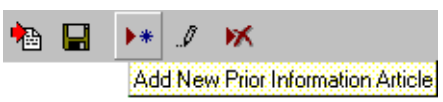
Assigning Prior Information

Often some independent information exists about the parameters that we wish to optimize. This information may be in the form of unrelated estimates or of relationships between parameters. When this information is included, it can lend stability to the parameter estimation process, especially when parameters are highly correlated. Correlated parameters can lead to non-unique parameter estimates because varying them in certain linear combinations may cause very little change in the objective function. In some cases, this non-uniqueness 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 non-uniqueness and provide stability.

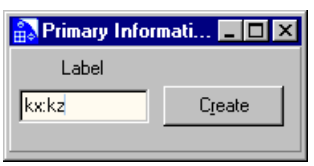
For a detailed description of how prior information is incorporated into the PEST algorithm, see Chapter 2 of the PEST Manual.

Prior information must be of a suitable type to be included. Both simple equality and linear relationships are acceptable. A weight must be included with each article of prior information. In theory, this weight should be inversely proportional to the standard deviation of the right hand side of the prior information equation. In practice, however, the user simply assigns the weights according to the extent to which he/she wishes each article of prior information to influence the parameter estimation process.

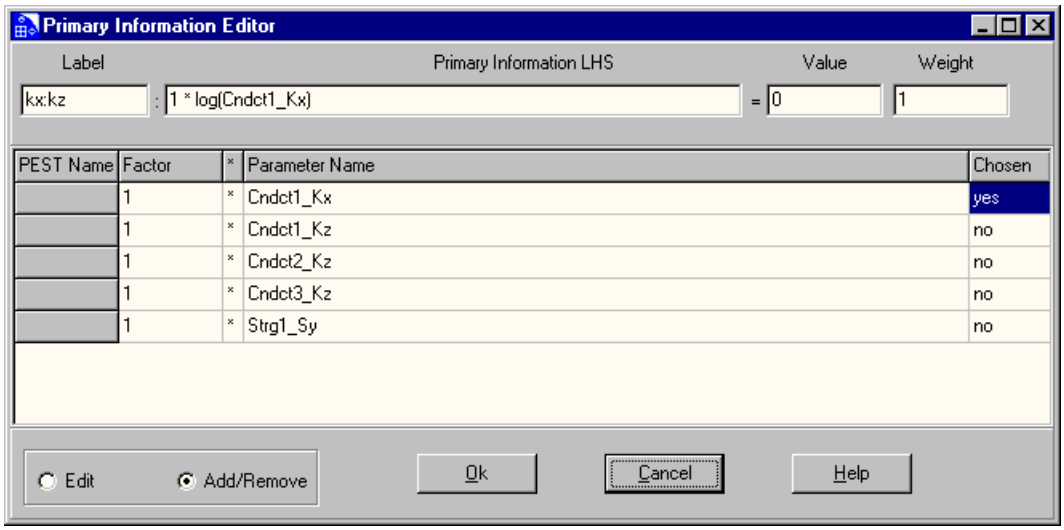
From the Prior Information tab a new prior information equation can be added by selecting the add item icon.



which brings up an initial dialogue asking for the name of the equation.

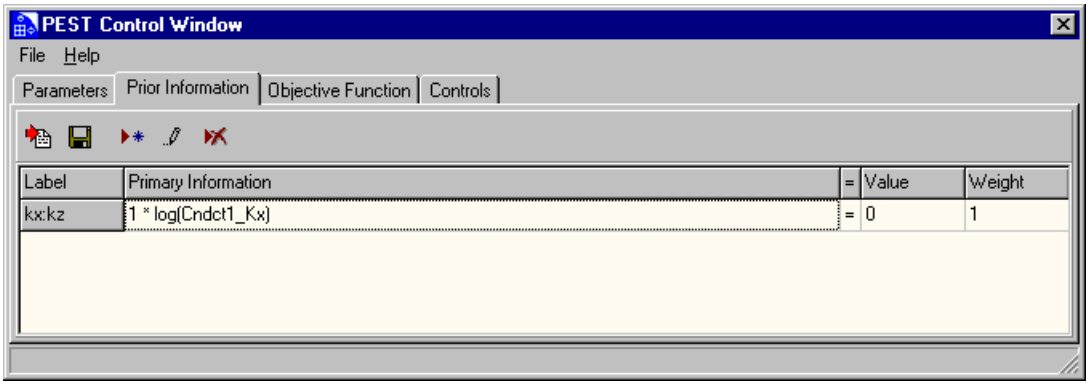


Once you have typed in a name, the following editor appears, which allows you to specify a simple linear equation for your prior information.



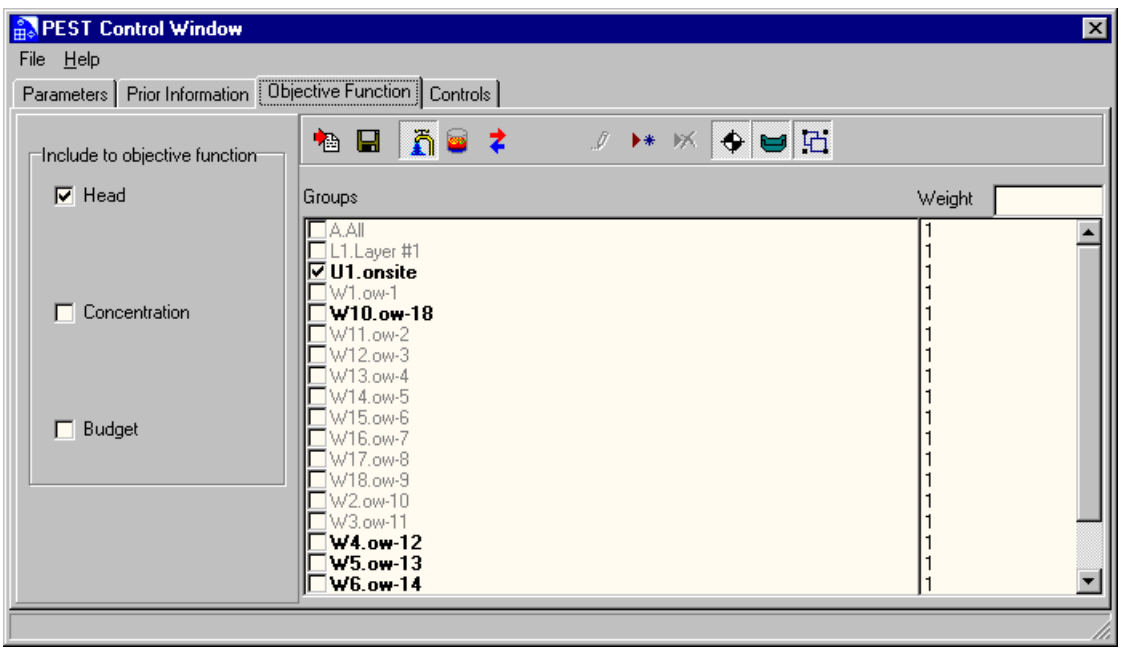
In this example, a 1:10 vertical conductivity ratio has been specified for Conductivity Zone 1. PEST will try to find the set of parameters that minimizes the objective function, while trying to keep the Kx to Kz ratio as close to ten as possible. The higher the weight, the greater will be the impact on the objective function. If you tied the Kx to Kz in the ratio of 10:1 then only Kx would varied by PEST and Kz would simply follow along during the optimization process. The ratio between Kx and Kz would be always 10:1. Using prior information in this way, gives PEST some flexibility in choosing a ratio that is close to 1:10.

Once the primary information has been defined the PEST Control Window will look similar to the following:

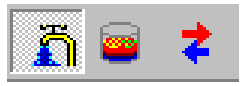


Assigning the Objective Function

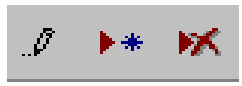
After deciding which Visual MODFLOW parameters to estimate, you must decide which observations will be used to calculate the objective function. Selecting the Objective Function tab will give you a list of all your available head, concentration or flow observations.



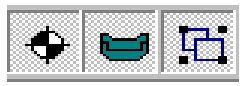
The three icons, shown here, allow you to display either head observations (from MODFLOW), concentration observations (from MT3Dxx) or flow observations (from Zone Budget).



On the right hand side are the group icons, which allow you to edit or delete a selected user-defined group or add a new user-defined group.



The three icons on the far right allow you to selectively display the well groups, the layer groups or the user groups.

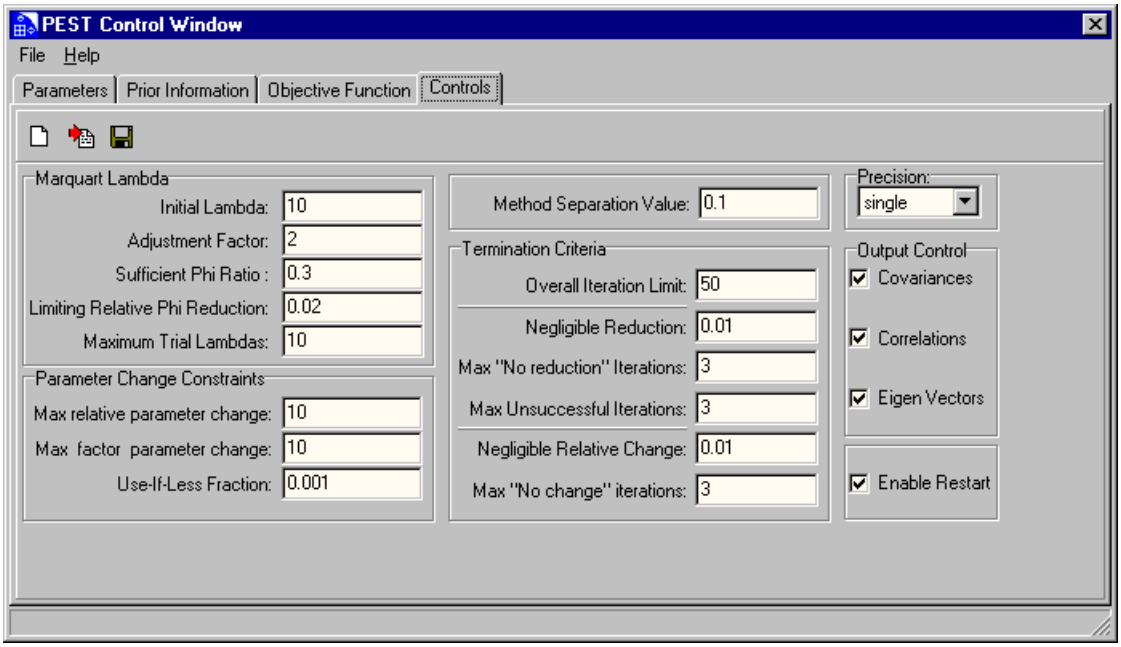


All the wells in the project are assigned to a group and PEST determines the contribution of each group to the overall change in the objective function. When you select a group, all the groups that intersect with that group are checked and then greyed out because PEST does not allow an observation to belong to more than one group. For example, if you select an individual well group and that well belongs to a layer group and a user-defined group, then the layer group and user-defined group will also be greyed out and cannot be selected.

Finally, you must select the appropriate checkboxes on the left side of the tab to tell Visual MODFLOW to include the head, concentration and flow observations when it translates the PEST files.

Controlling the PEST Run

The final tab in this dialogue is the Controls Tab. In this tab, you can modify any of the default values for the PEST controls. A brief description of these variables is provided below. More detailed descriptions of how some of these parameters are used by PEST can be found in Chapters 2 and 3 of the PEST Manual.



Marquardt Lambda

As outlined in on pag e18 of the PEST User's Manual, 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.

For high values of the Marquardt parameter (and hence of the Marquardt lambda) the parameter estimation process approximates the gradient 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.

Initial Lambda - RLAMBDA1

This is the initial real-value of the Marquardt lambda. An initial value between 1. and 10. is usually appropriate, though if PEST complains that the normal matrix is not positive definite, you will need to provide a higher initial Marquardt lambda.

Adjustment Factor - RLAMFAC

This is the real factor by which the Marquardt lambda is adjusted. It 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.

The first lambda that PEST uses is the lambda inherited from the previous iteration, reduced by the factor RLAMFAC (unless it is the first iteration, in which case RLAMBDA1 is used). Unless the objective function is reduced to less than PHIRATSUF of its value at the beginning of the iteration, PEST then tries another lambda, again reduced by the factor RLAMFAC. If the objective function is lowered but is still above PHIRATSUF of the starting objective function, PEST reduces lambda yet again. Otherwise PEST increases the first lambda in the iteration by the factor RLAMFAC. If the objective function begins to rise, PEST accepts the previous lambda and the corresponding parameter set and moves on to the next iteration.

Sufficient Phi Ratio - RHIRATSUF

During any one optimisation iteration first lowers lambda and, if this is unsuccessful in lowering the objective function, it then raises lambda. If it calculates an objective function, which is a fraction PHIRATSUF or less of the starting objective function for that iteration, PEST moves on to the next optimisation iteration.

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 on which the optimisation equations of Chapter 2 are based. If it is set too high, PEST may not be able to lambda such that its value continues to be optimal as the parameter estimation process progresses.

Limiting Relative Phi Reduction - PHIREDLAM

If a new/old objective function ratio of PHIRATSUF or less is not achieved as different Marquardt lambdas 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.

If the relative reduction in the objective function between 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.

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 find its optimal value. 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 a new iteration.

Maximum Trial Lambdas - 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, experience has shown that a value closer to 10 may be more appropriate than one closer to 5. This gives PEST more chance to adjust to the reduction in the number of parameters during the process.

Parameter Change Constraints

If there is no limit on the amount by which parameter values may change, 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 non-linear problems.

PEST provides the two real, input variables, RELPARMAX and FACPARMAX, which can be used to limit parameter adjustments. Any particular parameter can be subject to only one of these constraints (i.e. a particular parameter must be either “relative-limited” or “factor-limited” in its adjustments).

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. 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.

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. You must not 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.

If RELPARMAX and FACPARMAX are set too high, the estimation process may founder. If PEST seems to be making no progress in lowering the objective function and an inspection of the PEST run record reveals that some or all parameters are

undergoing large changes at every optimisation iteration, then it would be a good idea to reduce RELPARMAX and/or FACPARMAX. Another sign that these variables may need to be reduced is if PEST rapidly adjusts one or a number of parameters to their upper or lower bounds, and the latter are set far higher or lower than what you would expect the optimal parameter values to be. A further sign is if, rather than lowering the objective function, PEST estimates parameter values for which the objective function is incredibly high.

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 non-linearity, 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. (If necessary use the OFFSET to shift the parameter domain so that it does not include zero.)

Max relative parameter change - RELPARMAX

RELPARMAX is the maximum relative change that a parameter is allowed to undergo between optimisation iterations

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

$$(b_{i-1} - b_i)/(b_{i-1})$$

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.

Max factor parameter change - FACPARMAX

FACPARMAX is the maximum factor change that a parameter is allowed to undergo.

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

b_{i-1}/b_i if $| b_{i-1} | > | b_i |$, or

b_i / b_{i-1} if $| b_i | > | b_{i-1} |$

If parameter b is factor-limited, this factor change (which either equals or exceeds unity according to equation 2.4) 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.

Use-if-less Fraction - FACORIG

If, during the estimation process, a parameter becomes very small, the relative or factor limit to subsequent adjustment of this parameter may severely slow its growth back to higher values. Furthermore, for the case of relative-limited parameters which are permitted to change sign, it is possible that the denominator of the relative-limited equation above could become zero.

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 equations above.

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. Relative limitations are not 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.

Method Separation Value - 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 Chapter 2. You must inform PEST through the group variables FORCEN and DERMTHD which method to use for the parameters belonging to each parameter group. If you allow PEST to switch between forward and central-difference methods, the variable PHIRREDSWH tells PEST when to switch.

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".

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.

Precision - 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. For example, 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".

Termination Criteria

In addition to the termination controls available in this dialogue, PEST will terminate if

- the objective function goes to zero,
- the gradient of the objective function with respect to all parameters equals zero,
- the parameter upgrade vector equals zero, or
- all parameters are at their limits and the upgrade vector points out of bounds.

For more detailed information on the Termination Criteria, see pa ge33 of the PEST User's Manual.

Overall Iteration Limit - NOPTMAX

This is the maximum number of iterations.

Negligible Reduction - PHIREDSTP

If the objective function does not change by more than this amount for NPHISTP iterations, PEST will stop. A suitable value for PHIREDSTP is 0.01, for most cases.

Max "No reduction" Iterations - NPHISTP

This is the maximum number of iterations that PEST will perform, if the objective function has not changed by at least PHIREDSTP

For many cases, 3 is a suitable value NPHISTP.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. 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.

Max Unsuccessful Iterations - NPHINORED

If there is no reduction in the objective function, below its current minimum value, for this number of "unsuccessful" iterations, PEST will stop.

NPHINORED is an integer variable, where a value of 3 is often suitable

Negligible Relative Change - RELPARSTP

If the largest relative parameter change for all variables over NRELPAR iterations is less than this amount, PEST will stop.

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.

Max “No change” Iterations - NRELPAR

This is the maximum number of iterations PEST will perform if the largest relative parameter change for all variables is below RELPARSTP.

NRELPAR is an integer variable. A value of 2 or 3 is normally satisfactory.

Output Control - ICOV, ICOR, IEIG

After the optimisation process is complete, PEST writes some information concerning the optimised parameter set to its run record file. It tabulates the optimal values and the 95% confidence intervals pertaining to all adjustable parameters. It also tabulates the model-calculated observation set based on these parameters, together with the residuals (i.e. the differences between measured and model-calculated observations).

If you wish, PEST will write the parameter covariance matrix, the parameter correlation coefficient matrix and the matrix of normalised eigenvectors of the covariance matrix to the run record file.

The integer variables ICOV,ICOR and IEIG determine whether PEST will output the covariance, correlation coefficient and eigenvector matrixes respectively. If the relevant integer variable is checked (set to 1), the pertinent matrix will be written to the run record file. If it is not checked (set to 0), it will not be written.

Enable Restart - RSTFLE

This character variable is set by means of a check box. If it is checked, then “restart” is written to the control file. If it is unchecked, the value “norestart” is written.

If restart is selected, PEST will dump the contents of many of its data arrays to a binary file (*projectname*.RST) at the beginning of each optimisation iteration. This allows PEST to be restarted later, if execution is prematurely terminated. PEST will also dump the Jacobian matrix to another binary file (*projectname*.JAC) every time this matrix is filled.

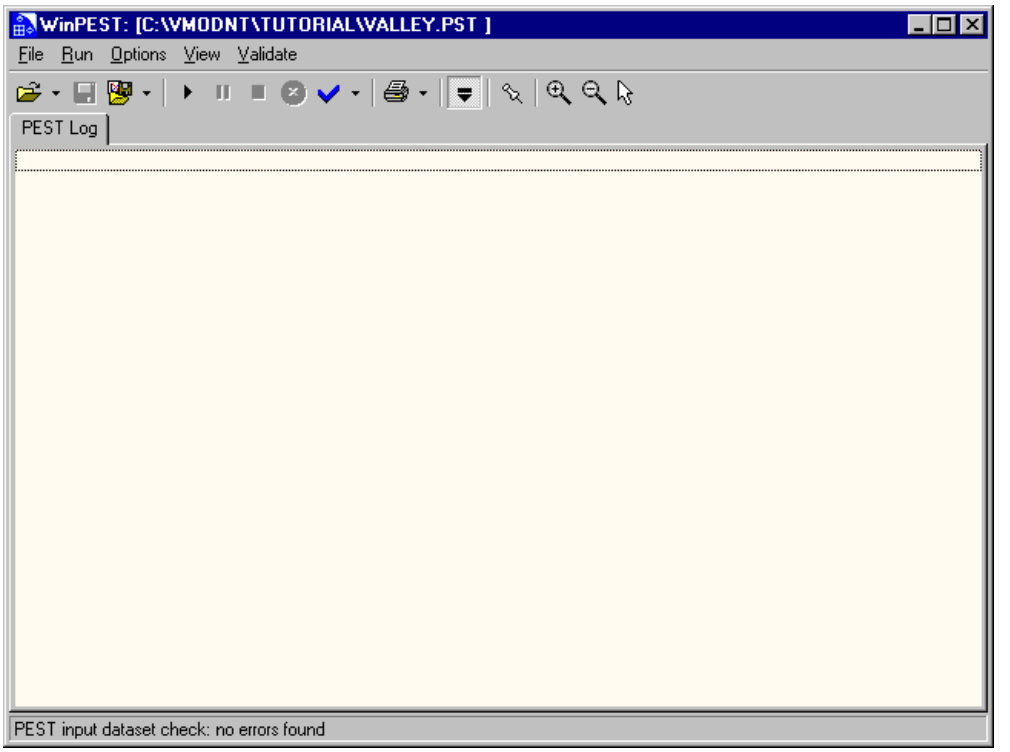
If restart is not selected, PEST will not intermittently dump its array or Jacobian data. Thus, later re-commencement of the optimisation is impossible.

Starting the PEST Run

To start the PEST run you must click on **[Run]** from the top menu bar in the Run Module. This following dialogue box will appear:

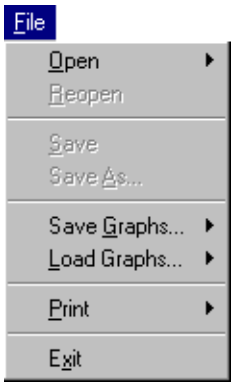
Now to launch PEST you must first select it from the list of available models. You must also select all of the numeric engines that you want PEST to run. Although, PEST can only estimate MODFLOW parameters at the moment, MT3D and Zone Budget must be run if you are using concentration and flow observations in your objective function.

Once the models have been selected, click on **[Translate & Run]** to create the PEST files and start WinPEST. Clicking on **[Translate & Run]** will translate the files and the following window will appear:



The top menu bar consists of the following options: **[File]**, **[Run]**, **[Options]**, **[View]**, and **[Validate]**. The drop-down **[File]** menu and/or the toolbar have the following features:

[Open] The **[Project]** option allows the user to select a PEST control file to open (***.PST**). The **[File]** option allows the user to open a selected PEST: control file (***.PST**), parameter hold file (***.HLD**), template file (***.TPL**), parameter file (***.PAR**), or instruction file (***.INS**).



[Reopen] Reopens a previous PEST control file.

[Save] Saves the current edit file with the existing name.

[Save As…] Save the current edit file with another name.

[Save Graphs] Saves PEST graphs to a series of files. Menu allows user to save graphs **[As Picture...]**, **[Current Plot]**, and **[All Plots].**

[Load Graphs] Loads PEST graphs. Menu allows user to load **[Current Plot]** or **[All Plots]**.

[Load/Save Plots] The button allows the user to **[Save Plots as...]**, **[Save Current Plot]**, **[Load to Current Plot]**, **[Save all Plots]**, and **[Load all Plots]**. The user is also able to select between **[Add Plots(s) (Load Style)]** or **[Overwrite Plots(s) (Load Style)]**, and the user is able to toggle on and off the option to **[Save Plots at End]**.

[Print] Allows the user to **[Print Current Tab]**, **[Multiprint Graphs]**, **[Print All]**.

[Exit] Exits the WinPEST window and returns to the Main Menu in Visual MODFLOW.

The drop-down **[Run]** menu and/or the toolbar have the following features:

[Run] Starts the PEST run. Menu allows the user to **[Start from Scratch]**, **[Restart last Iteration (/r)]**, **[Restart using last Jacobian (/j)]**.

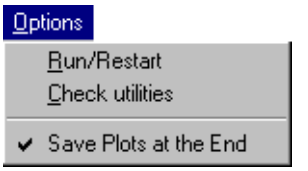
[Pause] Pauses PEST run.

[Stop with Statistics]Terminates PEST run with statistics.

[Stop without Statistics]Terminates PEST run with statistics.

[Check] Allows the user to check the **[All]**, the **[Instructions]**, or the **[Templates]**.

The drop-down **[Options]** menu has the following features:



[Run/Restart] Starts or restarts the PEST run.

[Check Utilities] Allows the user to select either to: ☑ **Generate model input**, or ☑ **Generate observation file**.

[Save Plots at the End]Allows the user to save the plots at the end of the run.

The drop-down **[View]** menu has the following features:

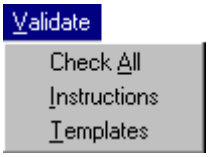
[Plots] Allows the user to select to view the following plots: Objective function, Composite Sensitivity, Parameters History, Marquardt Lambda, Calculated vs. Observed, Jacobian, Correlation, Covariance, and Eigenvectors.

[Log File] Selects the Log File tab as the top screen to view. This is the default tab.

[Run Record] Selects the Run Record tab as the top screen to view

[Hold Parameters] Displays the Hold Parameters Status screen. For more information on holding parameters please refer to pag e106 of this manual.

The drop-down **[Validate]** menu has the following features:



[Check All] Checks both the emplates and the instructions to the ***.PST** file for errors.

[Instructions] Checks the set of input instructions to the ***.PST** file for errors

[Templates] Checks the ***.PST** file templates for errors.

The remaining toolbar buttons on the WinPEST window have the following features:

[Autoscroll to the bottom of PEST log]If the Autoscroll button is depressed during the PEST run then the display in the WinPEST dialog box will show the laterst line of data output to the log file. If the toolbar button is not pressed then the user can scroll freely through the log file while WinPEST is running.

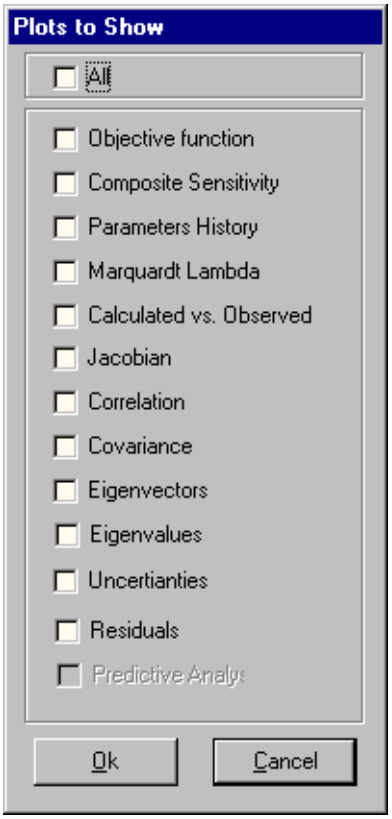
[Zoom In] Only available when the WinPEST plots are activated, the zoom in feature allows the plots to be examined more closely.

[Zoom Out] Only available when the WinPEST plots are activated, the zoom out feature allows the plots to be examined in less detail.

[Clear Zoom State]Toolbar option allows the user to return to the original scale of the plot they are viewing.

WinPEST Plots

WinPEST plots can be choosen by selecting **[View][Plots]**. The following dialogue box will appear:



Select the plots you wish to view by placing a checkmark in the correct box(es) and then press [OK]. A tab with the plot(s) respective name will be added to the WinPEST window. Select the tab to view the plot and. Notice on the top menu bar, the current plot name with its associated options will appear on the top menu bar when viewing the plot.

Objective Function

Composite Sensitivity

Parameters History

Marquardt Lambda

Calculated vs. Observed

Jacobian

Correlation

Covariance

Eigenvectors

Eigenvalues

Uncertainities

Residuals

Predictive Analysis

5

5 - Evaluating the PEST Run

The following chapter contains information on:

- The PEST Output Files and,
- The PEST Run Record.

PEST Output Files

Automatically PEST generates three default files. These are:

- the Parameter Value File,
- the Parameter Sensitivy File,
- the Residuals File,
- and, if specified, PEST will generate a number of Other Files.

The Parameter Value File

At the end of each optimization iteration, PEST writes the best parameter set achieved thus far (i.e. the set for which the objective function is lowest) to the parameter value file, *projectname*.PAR. At the end of a PEST run, the PEST parameter value file contains the optimal parameter set as seen below in Example 5.1.

```
single point
       ro1   1.0000001.0000000.000000
       ro2   40.000901.0000000.000000
       ro3   1.0000001.0000000.000000
        h1   1.0000031.0000000.000000
        h2   9.9997991.0000000.000000
```

Example 5.2: A parameter value file.

The first line of the parameter value file contains the character variables PRECIS and DPOINT, the values for which were provided in the PEST control file (see Appendix A). This is followed by a line for each parameter, each line containing the PEST parameter name, its current value and its SCALE and OFFSET.

The Parameter Sensitivity File

Most of the time consumed during each PEST optimization iteration is devoted to the calculation of the Jacobian matrix. Recall that each column of the Jacobian matrix lists the derivatives of all "model-generated observations" with respect to a particular parameter. During this process the model must be run at least as many times as the number of adjustable parameters.

Based on the contents of the Jacobian matrix, PEST calculates a figure related to the sensitivity of each parameter with respect to all observations, weighted according to the user-assigned weights. The "sensitivity" of parameter *i* is defined as:

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

where **J** is the Jacobian matrix and **Q** is the "cofactor matrix" which, in the present context, is a diagonal matrix whose elements are the squared weights of the observations. Thus, the sensitivity of the *i*'th parameter is the 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.

Immediately after it calculates the Jacobian matrix, PEST writes parameter sensitivities to a "parameter sensitivity file" called "*projectname*.SEN". Example 5.2, below is an extract from a parameter sensitivity file.

```
PARAMETER SENSITIVITIES: CASE VES4
OPTIMISATION ITERATION NO.  1 ----->
Parameter name Group   Current value    Sensitivity
resis1         resis   5.98563          16.5173
resis2         resis   103.493          9.58584
resis3         resis   23.4321          36.9477
thick1         thick   0.43454          9.44217
thick2         thick   13.4567          5.17165

OPTIMISATION ITERATION NO.  2 ----->
Parameter name Group   Current value    Sensitivity
resis1         resis   8.546532         9.20533
```

Example 5.3: Part of a parameter sensitivity file.

Information is appended to the parameter sensitivity file during each optimization iteration immediately following the 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.

This information on parameter sensitivity can be very useful when considering whether to hold various parameters during the estimation process.

The Residuals File

At the end of its execution, PEST writes the "residuals file", *projectnam* .RES, which lists in tabular form observation names, the groups to which various observations belong, measured and modeled observation values, differences between these two (i.e. residuals), measured and modeled observation values multiplied by respective weights, and weighted residuals. This file can be readily imported into a spreadsheet for various forms of analysis and plotting.

Other Output files

If requested specified, PEST will intermittently store its data arrays and Jacobian matrix in binary files named *projectname.*RST and *projectname*.JAC respectively. 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 both of them.

PEST uses a one-line file named PEST.HLT to communicate with the run record display utility, DECIDE.EXE, in the event of an interruption to PEST execution. Occasionally PEST also uses a small file named PEST.TMP for temporary bookkeeping. Neither of these files contain any useful information as far as the user is concerned.

The PEST Run Record

As PEST executes, it writes a detailed record of the parameter estimation process to the file *projectname*.REC. In this section the PEST run record is briefly described. However, an example Run Record file, which is discussed in detail, can be found in Appendix B.

The Input Data Set

PEST commences execution by reading all the 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", which means that a particular PEST input variable has no effect on the optimization process.

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 occur if the space allocated to this parameter in a model input file is insufficient for the degree of precision specified in the PEST control file.

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 optimization 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 optimization iteration.

At the end of each optimization iteration PEST records either two or three (depending on the input settings) very important pieces of information. These are the maximum factor parameter change and the maximum relative parameter change. As was discussed in Chapter 2, each adjustable parameter must be designated as either factor-limited or relative-limited. A suitable setting for the factor and relative change limits (i.e. FACPARMAX and RELPARMAX) may be crucial in achieving optimization stability. Along with the value of the maximum factor or parameter change encountered during the optimization 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 held temporarily fixed should trouble be encountered in the optimization process.

In addition to the current objective function value at the start of the optimization process and at the start of each optimization iteration, PEST also lists the contribution made to the objective function by each the observation groups and by all prior information. This is valuable information, for if a user notices that one particular group, or the prior information equations, are either dominating the objective function or are not “seen” because something else was dominating, the observation or prior information weights could be adjusted and the optimization process started again.

Optimized Parameter Values and Confidence Intervals

After completing the parameter estimation process, PEST prints the outcome to the third section of the run record file. First it lists the optimized 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 (e.g. 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 that was used to derive the equations for parameter improvement in each PEST optimization 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 the example in Appendix B.
- 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. 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 optimized 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 be small, but some of the parameters may be highly correlated. This will give rise to large covariance values (and hence large confidence intervals) for the correlated parameters.

With reference to the last point above, if several parameters are well correlated, then they can be varied in harmony such that when they are varied in a manner that 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 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.

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.

Note that at the end of a PEST optimization run a listing of the optimized parameter values can also be found in the PEST parameter value file, *projectname*.PAR.

Observations, Prior Information and Residuals

After it has written the optimized 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 optimized parameter set. The differences between the two (i.e. the residuals) are also listed, together with the user-supplied set of observation weights. Tabulated residuals and weighted residuals can also be found in file *projectname*.RES.

Following the observations, the user-supplied and model-optimized 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.

The Covariance Matrix

The parameter covariance matrix is written to the run record file, if you select this option in the Visual MODFLOW PEST Control Dialogue. 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 optimized 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 insofar 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. The variance of a parameter is the square of its standard deviation. The off-diagonal elements of the covariance matrix represent the covariances between parameter pairs.

If there are more than eight adjustable parameters, the rows of the covariance matrix are written in "wrap" form (i.e. after eight numbers have been written, PEST will start a new line to write the ninth number). Similarly if there are more than sixteen adjustable parameters, the seventeenth number will begin a new line. Note, however, that every new row of the covariance matrix begins on a new line.

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.

The Normalized Eigenvector Matrix and the Eigenvalues

PEST calculates the normalized eigenvectors of the covariance matrix and their respective Eigenvalues. The Eigenvector matrix is composed of as many columns as there are adjustable parameters, each column containing a normalized 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 semi-axis of the probability ellipsoid in *n*-dimensional adjustable parameter space.

If each eigenvector is dominated by a single element, then each adjustable parameter is well resolved by the parameter estimation process. However, where the dominance of eigenvectors is shared by a number of elements, parameters may not be well resolved. The higher the eigenvalues of those eigenvectors for which dominance is shared by more than one element, the less susceptible are the respective individual parameters to estimation.

6

6 - Troubleshooting PEST

The following section of the manual contains information on:

- Run-time Errors,
- MODFLO and MT3D Considerations,
- What to do if PEST Won't Optimize,
- Holding Parameters and,
- Restarting the PEST Execution.

Run-time Errors

Within the WinPEST interface, limited checking of the input data set can be done. However, if there is an error or inconsistency in the input data, PEST will terminate execution with a run-time error message. PEST will not continue reading the input data files 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 a run-time error message. This may happen in MODFLOW, for example, if a cell goes dry that contains an observation point. 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 WinPEST output window and the model output files (*.lst for MODFLOW and *.ot for MT3D) for a model-generated error messages.

A floating point or other compiler-generated error, followed by a PEST run-time error message, usually means that the model, not PEST, generated the error. You should then carefully inspect the model output files (*.lst for MODFLOW and *.ot for MT3D) 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 domain, in which case you will have to adjust their upper and/or lower bounds accordingly in the PEST control file.

Another model-related error, which can lead to a PEST run-time error of this kind, will occur if the path names in the PEST control files are wrong. This can occur if you change the path names in the control file or move the projects to a different directory. 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 WinPEST informs you through its dialogue 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 labeled as such, as shown below.

```
*****************************************************
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.
*****************************************************
```

Considerations for MODFLOW and MT3D

The most common cause of failure of PEST to optimise MODFLOW and MT3D parameters is poor settings for the PEST variables governing derivative calculations and inappropriate settings for the variables RELPARMAX and FACPARMAX.

A common cause of premature MODFLOW termination in a transient run is SOR, SIP or PCG2 convergence failure at a certain time step.

If there is no change in the objective function after several iterations, and PEST finally stops because "phi gradient zero" or something similar, then it is possible that MODFLOW or MT3D did not actually run, and that MODBORE or MT3BORE was reading an old head, drawdown or concentration file each time it ran the composite MODFLOW/MODBORE or MT3D/MT3BORE model. However, this is unlikely

unless you are using WinPEST outside of Visual MODFLOW since WinPEST and Visual MODFLOW, by default, delete the old files before starting a new PEST run.

If you suspect that MODFLOW or MT3D has not run, monitor the MODFLOW and MT3D output in the Win32 MODFLOW Suite dialogue during the PEST run.

If you try to optimize MT3D parameters based on a set of model-generated, theoretical "measurements", the optimization should work if the model-generated measurements were created using the same number of transport steps as MT3D uses when under the control of PEST. If the number of steps was different, PEST should still provide an optimal parameter set. However, this set may not be exactly the one that gave rise to the model-generated dataset in the first place because of slight differences in MT3D-calculated concentrations with differences in transport step size. PEST will, in fact, obtain a parameter set for which the concentrations calculated on the basis of the transport step size used for optimisation agree as closely as possible with those calculated on the basis of the transport step size used to generate the theoretical data.

Parameter Transformations and Bounds

PEST allows you to logarithmically transform adjustable parameters during the parameter estimation process. For some parameters this can hasten the optimisation process considerably. For other parameters it can slow it down. When calibrating MODFLOW models the log transformation of hydraulic conductivity, transmissivity, inter-layer leakance and storage coefficient can have a positive effect on estimation speed and stability. However, recharge is better left untransformed. While the situation is not as clear with MT3D, it appears that dispersivity estimates converge faster when logarithmically transformed. No clear recommendation can be made for other parameters. However, trial and error with your particular problem should soon provide the answer.

Appropriate parameter upper and lower bounds is also important to the success of the optimisation process. If a parameter is log-transformed its lower bound must be greater than zero. Also, realistic bounds should be placed on MODFLOW parameters, such as storage coefficient, for which there are physical limits to the range of allowable values.

When undertaking parameter estimation using MT3D, parameter bounds can be critical. MT3D parameter values can influence the size of the time step used by MT3D, unless the user specifies that a suitably small transport step be used regardless of any parameter value. This latter specification results in MT3D using the same number of transport steps from model run to model run. For a given flow field this, in turn, enforces accurate derivatives calculation, rapid estimation convergence and optimisation stability. However if certain estimated parameters are allowed to take on high enough values and others are allowed to take on low enough values in the course of the optimisation process, MT3D will override the user-specified transport step size, choosing an appropriately small step size of its own, thus breaking the step size consistency between model runs. It is important to prevent this from happening by

restricting parameter variation to a realistic range through the designation of suitable upper and lower parameter bounds.

Dry Model Cells

If a layer is unconfined and the water level in a cell falls beneath the cell bottom, MODFLOW declares the cell as dry. If the BCF1 package is used, the cell stays dry forever. However, the BCF2 package (and later BCF packages) allows dry cells to re-wet depending on the water levels in neighbouring and underlying cells.

The occurrence of dry cells in a simulation can have undesirable consequences, particularly if the BCF1 package is used. It often leads to a "cascading" effect in which the drying of one particular cell prevents water inflow to a downstream cell, which then becomes dry itself and so on.

The drying (and re-wetting) of cells can have a disastrous effect on MODFLO parameter estimation because, no matter which BCF package is used, model outcomes are no longer continuous with respect to adjustable parameters. This is because a small parameter change may result in certain heads crossing a "threshold" (e.g. the aquifer base or re-wetting level) at which a significant and discontinuous change in local aquifer flow conditions takes place. Furthermore, if an observation points lies within a dry cell, MODBORE is unable to calculate a head for that observation point, writing "dry_cell" to the heads column of its output file instead of the head for that borehole. When PEST reads the file, its inability to read a number where it expects to find one causes a run-time error.

There are two ways to ensure that cells do not dry out and cause problems for PEST. The first is the easiest and involves setting of parameter upper and lower bounds so tightly that no parameter is allowed to stray into an area where it causes a model cell to go dry. The second method, which is only suitable for single layer models, is to make a small adjustment to the MODFLOW source code and re-compile it. Visual MODFLOW includes a version of MODFLOW with this change.

In the BCF2 package, the following lines of subroutine SBCF2H have been changed from:

```
C6------CHECK TO SEE IFSATURATED THICKNESS IS GREATER THAN ZERO.
      IF(THCK.LE.0.) GO TO 100
```

to:

```
C6------CHECK TO SEE IFSATURATED THICKNESS IS GREATER THAN ZERO.
      IF(THCK.LE.1.0) THCK=1.0
```

With the above alteration, the calculated head in an unconfined layer is allowed to drop continuously below the base of the aquifer (the fact that it is below the aquifer base can be detected while contouring the results). However, the thickness of water is not allowed to drop below a certain lower limit, in this case one length unit (adjust this limit

to suit yourself). Because the water thickness never becomes negative, MODFLOW never declares the cell as dry.

It could be argued that this alteration leads to an impossible situation whereby a cell's water level is below its base yet the transmissivity of that cell is the same as if the cell contained one length unit's depth of water. However, in many single-layer models this is not such a bad assumption. If model cells are large, it may be unlikely that the entire cell area would dry out. Only those parts of the cell with a higher-than-average botto elevation would dry out. The cell would still be able to transmit water to neighbouring areas, albeit with a reduced capacity. Furthermore, keeping cells wet in this manner may degrade model performance to a lesser extent than the even more unrealistic cascading of dry cells.

In the case of multi-layer models, the above modification tends to the model unstable and impedes convergence.

Optimising Parameters for MODFLOW and MT3D Together

Currently, Visual MODFLOW is set up to optimize only MODFLOW parameters, but MT3D concentrations can be included in the objective function. In this case, Visual MODFLOW will create instruction files for reading MODBORE and MT3BORE output files and a single PEST control file which includes all borehole measurements. The assignment of a suitable weighting between water level and concentration measurements needs to be established.

Some of the pitfalls of simultaneous MODFLOW/MT3D optimisation have been explained in Section “Derivative precision in MT3D” on page 9 9 . Nevertheless, if you think that this would assist the calibration of your particular model, a PEST run can easily be set up to do it.

Alternatively, if you want to also estimate MT3D parameters, after translating the PEST files, you can modify the PEST Control file to include the MT3D parameters and create the necessary template files for the MT3D input files (see Appendix B). In this case, the "model" called by PEST will run MODFLOW and MODBORE followed by MT3D and MT3BORE. However the use of this simple model can lead to unnecessary MODFLOW runs in the derivatives calculation phase of each optimisation iteration, for while an increment to a MODFLOW parameter value will have an effect on concentrations calculated by MT3D, the inverse is not true; an alteration to a MT3D parameter will have no effect on MODFLOW-calculated heads or drawdowns. Hence when an MT3D parameter is incrementally varied there is no need to run MODFLOW prior to running MT3D.

If you have the unrestricted version of WinPEST, that comes with Visual PEST, you can circumvent this problem in the “batch” model run by PEST. For example, let us assume that PEST is optimising transmissivity for MODFLOW simultaneously with dispersivity for MT3D. The transmissivity array is located in the MODFLOW input file BCF.DAT. Accordingly a template named BCF.TPL is constructed for that file.

However PEST is informed that the model input file corresponding to BCF.TPL is, in fact, a file named BCF.HLD, a temporary "holding" file. Prior to running the model, PEST writes the model input files BCF.HLD and DSP.DAT, the latter holding the MT3D dispersivity array. If BCF.HLD differs from the previous MODFLOW input file BCF.DAT, then BCF.HLD is copied to BCF.DAT and MODFLOW is run (with appropriate safeguards against MODFLOW convergence problems). However if BCF.DAT and BCF.HLD are identical, there is no use in running MODFLOW. In this case execution of the batch process is taken to label1 where MODBORE is run. (The running of MODBORE is necessary because, prior to actually running the model, PEST deletes any model output files that it must later read. In this way PEST will know if, for any reason, the model failed to run; it also obviates the possibility of inadvertently reading a model output file produced on a previous model run.) MT3D, followed by MT3BORE, is then run irrespective of whether MODFLOW has been run or not.

If PEST Won't Optimize

WinPEST allows you to closely follow the progress of an optimization run through its dialogue and graphical output. By watching the value of the objective function, you can monitor PEST's ability and efficiency in lowering the objective function. 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. You may re-commence PEST execution where the Jacobian matrix was last calculated to re-compute the last parameter upgrade vector, or you can continue execution with the selected parameters held fixed for a while.

The first time you optimize a model, 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 for which you have field measurements, and then use these fabricated observations in place of your field data. Then run PEST using, as your initial parameter estimates, the parameters from 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 round-off 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 data set for which you know PEST should achieve a low objective function value, that you can adjust PEST control variables to tune PEST to the model. It is unlikely that the objective function will be zero. Although, depending on the number of observations and their magnitudes and weights, the objective function should be as close to zero as round-off errors will permit. In most cases, PEST is able to

solve a parameter estimation problem using substantially less than 20 optimization iterations.

If PEST does not lower the objective function, or lowers it slowly, the following two sections outline some of the reasons that PEST may perform poorly. In most instances the problem can be rectified.

Obtaining Sufficient Precision of the Derivatives

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

Try not to make parameter increments too large, 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.

Experience in calibrating MODFLOW has shown that it is best to calculate derivatives using relative rather than absolute increments (i.e. the PEST derivative control variable INCTYP is set to "relative"), and that a value of between 0.01 and 0.05 is suitable for DERINC. However, for safety's sake, it is wise to back this up with an appropriate value for DERINCLB, i.e. the absolute increment lower bound. For MT3D calibration DERINC is best set to 0.05 or higher if using a MOC scheme. For both MODFLO and MT3D, FORCEN should be set to "switch" while values of 2.0 and "parabolic" are suitable for DERINCMUL and DERMTHD in most cases. If you undertake a dummy run using model-generated "field data", the best values for these variables for your particular case will soon become apparent.

The estimation of recharge is a special case. Recharge can vary greatly over a model domain. Also, for some models, it may take on negative values. It has been found that an INCTYP setting of "rel_to_max" is often suitable for recharge parameters, and that a suitable value for DERINC is, again, 0.01 to 0.05. Remember recharge parameters should not be log-transformed.

Derivative Precision in MODFLOW

In Visual MODFLOW, you can select from among the SSOR, SIP, PCG2 and WHS solvers. For all of these methods, convergence is achieved when the maximum head change between successive solutions is less than a user-defined threshold, HCLOSE. The PCG2 method also requires the user to supply a convergence threshold for its inner iterations (RCLOSE), however, it is HCLOSE, not RCLOSE, that determines solution precision.

The lower HCLOSE is set, the higher the precision calculated. Thus, HCLOSE should be set low when using MODFLOW with PEST. A value of about 10^{-4} or less is recommended. However, because MODFLOW then requires more iterations to converge to this tighter convergence criterium, the maximum number of iterations (outer iterations for PCG2 and WHS solvers) should be increased.

With HCLOSE set this low, solution convergence may not always be achieved within the maximum number of iterations. In fact, for some parameter sets, solution convergence to may never be achieved even with the maximum number of iterations set very high. Unfortunately, for transient simulations, when MODFLOW fails to converge within the maximum number of iterations for a particular time step, it will abort execution instead of moving on to the next time step. This is disastrous for PEST. If MODFLOW aborts, not all the head arrays expected by MODBORE will be written. Then when MODBORE runs following MODFLOW, it will fail and not write its output file. When PEST tries to read the MODBORE output file, it will abort with an error message that it is unable to find the .HOB file.

There are two ways to overcome this problem. The easiest way is to make a slight alteration to the MODFLOW source code. In the MAIN program unit, under the comment labelled "C7C6", delete or comment out the line:

```
IF(ICNVG.EQ.0) STOP
```

With this modification, MODFLOW will continue onto the next time step whether solution convergence was achieved or not. Visual MODFLOW includes a version of MODFLOW that has incorporated this change.

The second option, which is available if you have the unlimited version of WinPEST, is to insert a little intelligence into the batch file which PEST calls as the model. MODBORE requires the user to inform it how many arrays to expect in the .HDS file that it will read. If the number of arrays in this file differs from what it expects, it will abort with a DOS errorlevel setting of 100. A statement in the batch file immediately following the MODBORE command can be used to trap this error event. Usually, most appropriate action would be to substitute a new MODFLOW-solver-package input file, containing a higher value for HCLOSE, for the one which MODFLOW has just read, and then re-run MODFLOW. If MODFLOW fails to converge again, another file can be substituted in which HCLOSE is set even higher. Eventually MODFLOW will converge and MODBORE will be able to complete its run. It is important to note, however, that this procedure will not work if the solver package file contains any adjustable parameters. For example, if you were using the PCG2 solver, you could copy PCG2.DAT (created by Visual MODFLOW) to PCG2.H1, PCG2.H2, PCG2.H3, PCG2.H4 and PCG2.H5. In each of these files increase the value of HCLOSE by say a factor of two. Make sure, however, that the initial MODFLOW run uses the tightest solution convergence criterion.

Derivative precision in MT3D

Depending on which version of MT3D you use and what options you specify, MT3 may or may not use an iterative solver to calculate solute concentrations. In MT3DMS-based versions of MT3D (MT3DMS and MT3D99) you have the option of selecting implicit or explict solution procedures for the finite-difference methods. If the MOC, MMOC or HMOC schemes are used, MT3D moves particles through the model domain to solve the advection component of the solute transport equation. The dispersion, source-sink mixing and chemical reaction components are solved using either an explicit or implicit finite-difference technique depending on the version of MT3D.

Whereas the explicit solution scheme presents no problems for solution precision, the particle tracking methods can pose problems for the accuracy of derivatives calculated with respect to adjustable parameters. While solute concentrations calculated by MT3D are precise enough for ordinary usage, a significant loss of precision occurs when the outcomes of subsequent runs, are subtracted from each other to calculate derivatives.

The problem stems from the fact that there is only a finite number of particles. This introduces "thresholds" throughout the model domain. For example, the number of particles in a cell at the end of a time step will depend on the flow regime calculated by MODFLOW. A slight change in, for example, the transmissivity, can alter the number of particles in a cell from, say, 9 to 10. This, in turn, will result in a discontinuous change in the solute concentration calculated for that cell as the transmissivity is slightly varied. Therefore, it will be difficult to accurately calculate the derivative of the concentration in that cell with respect to transmissivity if the transmissivity is an adjustable parameter.

The obvious solution is to not calibrate MODFLOW and MT3D together if a particle tracking scheme is used for solute transport. If MODFLOW is first calibrated using borehole head data, and then MT3D is calibrated separately using measured borehole concentrations, the flow regime by MT3D will be constant during the calibration process. Although this is a step in the right direction, it is still not sufficient to circumvent the problem of MT3D output "granularity" when using particle tracking methods.

MT3D updates solute concentrations at time intervals known as "transport steps". When using the explicit finite-difference or TVD schemes, each transport step must be small enough such that none of the stability criteria pertaining to the different components of the overall transport equation are violated. There is a stability criterion associated with the advection term, the dispersion term, the source-sink mixing term, and the chemical reaction term. All these criteria depend on system properties. Hence if one or more of these properties is being estimated by PEST, and thus changes from run to run, so too will one or more of the stability criteria. If MT3D is allowed to select the transport step size itself based on the tightest of the various stability criteria which it must meet (as it does when the input variable DT0 is set to zero or negative), then the transport step size may vary from run to run as adjustable parameters are varied. This,

in turn, will introduce model output "granularity" as it again becomes possible for the number of particles within a certain cell to vary slightly at a certain simulation time from one model run to the next.

Fortunately this problem is easily overcome. MT3D allows the user to select the transport step size through the input variable DT0 (if it is set positive). However MT3D overrides this choice if it fails to fulfil all of the model stability criteria. Hence, for consistency between model runs, DT0 must be chosen low enough that it will not be "undercut" at any stage of the optimisation process as PEST varies MT3D parameter values from run to run as it attempts to optimise them. While setting DT0 low in this manner results in an increased MT3D execution time, it does guarantee good PEST performance. On the other hand, if DT0 is set to a negative value, this value is used by MT3D regardless of the stability criteria. This has the danger though that some of the runs may produce unpredictable results.

As a complementary measure, it is important to place suitable bounds on adjustable MT3D parameters. For example, if you are estimating dispersivity and, in the course of the parameter estimation process, a dispersivity value becomes too high, the stability criterion associated with the dispersion term could necessitate a transport step size lower than your chosen DT0 value. In such a case MT3D will undercut DT0 in assigning the transport step size, with the result that the number of transport steps will vary between subsequent MT3D runs.

A further important rule to follow in order to maintain consistency in the movement of particles between cells from model run to model run, is that particles must be placed in a fixed pattern within a model cell rather than in a random pattern. You may also need to use more particles than normal to reduce mass balance discrepancies in diverging/ converging flow fields. Similarly, if solute source concentration is being estimated, care must be taken in assigning values to the variables NPL and NPH. In some cases, it may be advisable to set NPL equal to NPH.

Another "threshold" in MT3D that has the potential to introduce inconsistency between model runs involves the use of the MT3D input variable DHMOC used in the HMOC solution method to switch between the MOC and MMOC schemes. Experience has shown, however, that this does not cause too many problems, as borehole measurements used for model calibration tend to be in areas where solute concentrations are high and where the MOC, rather than the MMOC, scheme is operating. Conversely, in areas where the MMOC scheme is being used, the solute concentration is generally low. Thus, the contribution of a measurement residual from borehole in this area to the objective function is low, as long as the observation weight is not high. This further diminishes the potential for instability. Nevertheless, if PEST is having difficulty in optimising MT3D parameters and you are using the HMOC method, it may be worth attempting a calibration using the MOC scheme only.

Alternatively, dispense with particle-based schemes altogether, using the TVD or finite-difference methods to solve the advective component of the solute transport equation.

High Parameter Correlation

There is often a temptation in fitting models to data, to improve the fit between modeled 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 the model's ability to make reliable predictions. A high number of parameters may not represent a valid interpretation of the data set to which the model's outcomes are matched. Furthermore, as the number of parameters requiring estimation is increased, PEST's ability to lower the objective function by adjusting the values of these parameters is diminished due to round-off errors. This is particularly true for highly nonlinear models and applies not just to PEST but to any parameter estimation package.

The trouble with increasing the number of parameters is that, sooner or later, some parameters become highly correlated. With a high number of parameters, PEST may not be able to distinguish between different combinations of parameter values because various combinations can give equally low values of the objective function. As discussed in the previous chapter, the extent to which parameter pairs and groups are correlated can be determined from the correlation coefficient and eigenvector matrices.

If parameters are too highly correlated the matrix $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ of equation (8.18) 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 8.20), making it no longer singular, an upgrade vector will nevertheless be obtained. Eventually, unless circumvented by round-off errors, an objective function minimum will be obtained through the normal iterative optimization process. However the parameter set determined on this basis may not be unique. Furthermore, for highly nonlinear models, the objective function may have attained a local, rather than global, minimum. 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 non-uniqueness problem, the optimization 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 to fill the Jacobian matrix during each optimization 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 Chapter 8, using a high Marquardt lambda is equivalent to using the gradient optimization method. However the gradient 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. If lambda cannot be lowered because the normal

matrix become singular, or at best ill-conditioned, due to the excessive number of parameters requiring estimation, there will be no way to prevent this.

These troubles are often compounded by the fact that as parameter numbers are increased, each parameter may have a smaller effect on fewer observations. Hence the accuracy of derivatives calculation will suffer and, with it, PEST's ability to find the global objective function minimum in parameter space.

Note that the incorporation of prior information into the estimation process can often add stability to an over-parameterized 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.

Inappropriate Parameter Transformation

PEST allows adjustable parameters to be either log-transformed or untransformed. Log transforming appropriate parameters 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 as to which parameters are best log-transformed. However experience has shown that parameters, such as conductivity, whose values vary by one or more orders of magnitude often benefit from log transformation. Log-transformation of these parameters will often linearize the relationship between the parameters and the observations. Consequently, to the linearity assumption upon which the equations of Chapter 8 and 9 are based will be more valid.

The use of a suitable scale and offset may change the domain of a parameter such that logarithmic transformation becomes possible. The use of parameter scaling and offset is discussed in Chapter 9.

Highly Non-linear Problems

If the relationship between parameters and observations is highly nonlinear, the optimization process will be difficult. Such nonlinearity may be circumvented through logarithmically transforming some parameters, with or without a suitable offset and scaling factor. However, sometimes, log-transformation 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 round-off 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.

For all types of parameter estimation problems, but particularly for highly nonlinear problems, the closer user-supplied initial parameter values are to optimal parameter values, the greater the chance of PEST being successful.

Discontinuous Problem

The Gauss-Marquardt-Levenberg algorithm, on which PEST is based, assumes that the observations are continuously differentiable functions of the parameters. If this assumption is violated, PEST will have extreme difficulty in estimating parameters for the model. Although, it may have some success if the dependence is continuous, if not continuously differentiable.

Parameter Change Limits Set Too Large or Too Small

As outlined above for highly nonlinear problems, suitable relative and factor parameter change limits may allow an optimization in difficult circumstances. However if the change limits specified by RELPARMAX and FACPARMAX are too small, minimization of the objective function may be hampered as the upgrade vector is continually shortened. Inspecting the run record should reveal whether parameter upgrades are being limited by these variables. If the maximum relative or factor parameter changes per optimization iteration are consistently equal to the user-supplied limits, then you might want to increase these limits. However, if your model is highly nonlinear or "messy", it may be better to keep RELPARMAX and FACPARMAX low as this may prevent "overshooting" during the parameter adjustments.

You should exercise caution in choosing which parameters are relative-limited and which are factor-limited. Remember 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 greater than or equal to one, it can be reduced right down zero in a single step without transgressing the limit. This may cause parameter "overshoot" problems for some nonlinear models and you may want to consider a factor limit. However, a factor limit cannot be used if the parameter can change sign. This can be overcome by using an appropriate OFFSET to shift the parameter domain such that it does not include zero.

Finally, the offending parameter can be held at its current value, if the parameter adjustment vector is dominated by a particular insensitive parameter, such that the parameter is equal to its RELPARMAX or FACPARMAX limit and the changes to other parameters are minimal.

Poor Choice of Initial Parameter Values

In general, the closer the initial parameter values are to the optimal values (i.e. the values for which the objective function is at its global minimum), the faster PEST will converge to that global minimum. Not only can the initial parameter values reduce the run time of PEST, it may also make optimization possible. This is especially true for highly nonlinear models or models for which there are local objective function minima. It is important to remember that a little time spent in this trying to estimate independently a reasonable parameter set could be rewarded in greatly improved PEST performance.

Observations are Insensitive to Initial Parameter Values

In some models, the calculated values at the observation locations may be insensitive to the initial parameter values if the initial parameter values are poorly chosen. For example, if the layer conductivities in a multi-layer model are all set to the same initial value, even though there are several aquitards, the observations may be insensitive to the initial conductivities in the lower layers.

Alternatively, a parameter may have little effect on model outcomes at low values, yet a much greater effect at higher values. 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 the parameter's initial value should be within the sensitive part of its domain.

Poor Choice of Initial Marquardt Lambda

Typically, PEST will find its way to a near-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 an optimal lambda may not be successful. 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 is poor, these alternative lambdas may be little better than the first one, in terms of lowering the objective function. Based on the PEST parameters PHIREDLAM and NUMLAM, PEST may soon move on to the next optimisation iteration, after having achieved little in lowering the objective function. If this is repeated during subsequent optimisation iterations, PEST will soon terminate execution in accordance with one of its termination criteria.

In most cases, an initial Marquardt lambda between 1.0 and 10.0 works well. Nevertheless, if PEST spends the first few optimisation iterations significantly raising or lowering lambda, before achieving a lowering of the objective function reduction, then you may want to re-evaluate the initial lambda in subsequent PEST runs. If the

parameter estimation process simply does not "get off the ground", you should start over 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, you can increase the input variable RLAMFAC. However, it is not a good practice to keep RLAMFAC high throughout the optimisation run. If after increasing RLAMFAC PEST finds a lambda which seems to work, terminate PEST execution, supply that lambda as the initial lambda, reset RLAMFAC to a reasonable value (e.g. 2.0) and start the optimisation process again.

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

Upgrade Vector Dominated by Insensitive Parameters

Where many parameters are being estimated and some are far less sensitive than others, it is not uncommon to encounter problems in the parameter estimation process. PEST calculates an upgrade vector in which the insensitive parameters are adjusted by a larger amount relative to more sensitive parameters. Such adjustment of the insensitive parameters is necessary to ensure that they affect the objective function. However, the adjustment to any parameter is limited by the PEST control variables RELPARMAX and FACPARMAX. PEST reduces the magnitude of the parameter upgrade vector such that no parameter change exceeds these limits. Unfortunately, an insensitive parameter may dominate the parameter upgrade vector, restricting the magnitude of the upgrade vector. In this case, the change in the value of the insensitive parameter will be limited by RELPARMAX or FACPARMAX (depending on its PARCHGLIM setting) and will result in much smaller changes to other, more sensitive, parameters. Thus, the objective function may be reduced very little, if at all.

PEST records the names of parameters that have undergone the largest factor and relative changes at the end of each optimisation iteration. The problem is easily recognised when either the maximum relative parameter change or the maximum factor parameter change is equal to RELPARMAX or FACPARMAX respectively, and the objective function is reduced very little. More often than not, an inspection of the parameter sensitivity will reveal that these same parameters also possess a low sensitivity.

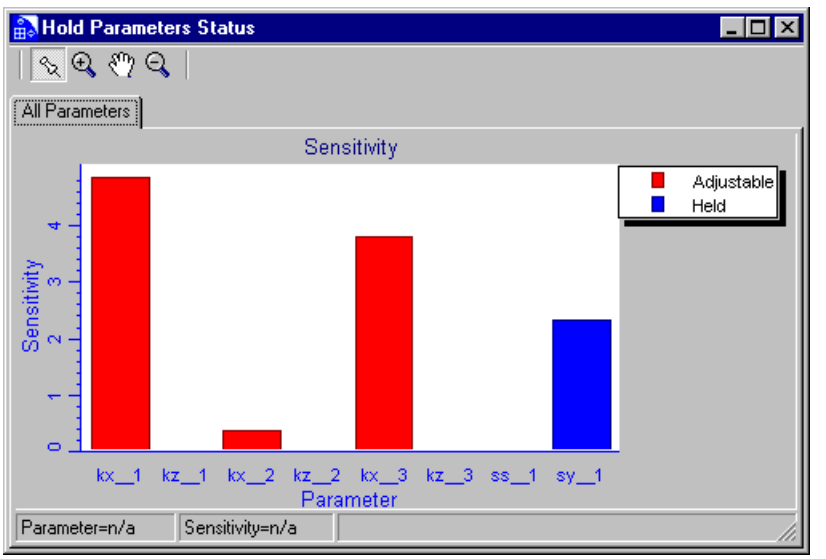
Under these circumstances, increasing RELPARMAX and FACPARMAX will not necessarily solve the problem. Parameter change limits are necessary to avoid unstable behaviour in the face of problem nonlinearity (this being the norm rather than the exception).

The solution is to hold insensitive parameters at their current values. In this way, PEST can often achieve a significant improvement in the objective function. Held parameters can then be released later in the parameter estimation process.

It may be that quite a few parameters need to be held in this manner. For example, once a particular troublesome parameter has been identified and held, another insensitive parameter may in turn dominate the parameter upgrade vector. This can continue until the set of parameters has been reduced to a set of sensitive parameters. Now, once the objective function has been reduced, the held parameters can be released one at a time until the final optimized solution has been found. Alternatively, you may prefer to preemptively hold all the parameters at once that are suspected to be insensitive.

Holding Parameters

In Visual MODFLOW the thumb-tack button allows you to interactively hold parameters during the parameter estimation process. Clicking on the hold icon brings up the following dialogue.



This dialogue contains a bar graph with a bar for each adjustable parameter. Simply double clicking on a bar in the bar graph will hold a parameter. Double clicking again will release the parameter. Once the parameter has been held, you can view the graphics and watch the optimization process in the main PEST dialogue

The Parameter Hold File

You will not normally deal with the hold file, since it is automatically generated by Visual MODFLOW. However, PEST allows some additional flexibility in the hold file that is not directly supported by Visual MODFLOW.

After it calculates the Jacobian matrix, and immediately before calculating the parameter upgrade vector, PEST looks for a file named "*projectname*.HLD" 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 it for the current optimisation iteration. You can edit the hold file at any time and PEST will read the file at the next opportunity.

Part of a parameter hold file is shown below:

```
relparmax 10.0
facparmax 10.0
lambda 200.0
hold parameter thick1
# hold parameter thick2
hold group conduct < 15.0
hold group thickness lowest 3
```

Entries in a parameter hold file can be in any order. Any line beginning with the "#" character is ignored and treated 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 to the run record file any alterations to its behaviour based on the hold 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 above. That is, 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". Otherwise, 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 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 line the example above. The format is as follows

`hold parameter` *`parnme`*

where *parnme* is the name of the parameter in the .PST file. If the parameter name is incorrect, PEST simply ignores the line. If the 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 sixth line in the example above illustrates how to hold all the parameters in a group. In this case, the format is

`hold group` *`pargpnme`* `<` *`x`*

where *pargpnme* is the name of a parameter group and *x* is a positive number that tells PEST to hold any parameter with a sensitivity less than *x*. Held parameters can be released by reducing *x* (to zero if desired), by deleting this line from the parameter hold file, or by deleting the parameter hold file itself.

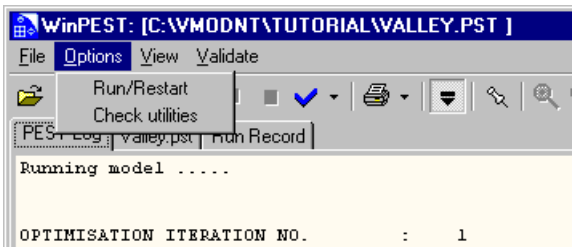
Finally, the seventh line in the example above shows how to hold the *n* most insensitive parameters in a particular parameter group. The format for this operation is

`hold group` *`pargpnme`* `<` *`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.

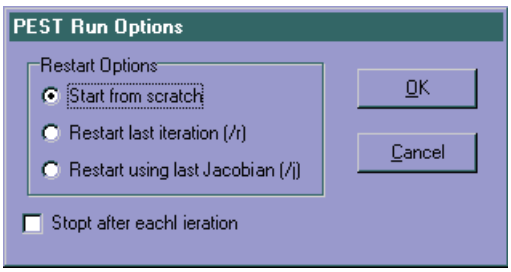
Re-starting PEST execution

Often PEST will run to completion with the set of parameters and values that you have specified. However, you may want to start and stop the parameter estimation process to adjust some of the PEST variables. PEST allows you to halt execution at any time and restart it again either at the end of the last iteration or using the last Jacobian matrix. This is very important because the calculation of the Jacobian matrix is a very time consuming operation. PEST stores the Jacobian matrix in a file each time it is calculated. The Jacobian matrix is then retrieved from disk if PEST is asked to re-calculate the parameter upgrade vector. Since, the size of the Jacobian matrix is determined by the number of parameters, if you want to remove a model parameter from the process, then you can either hold it constant and allow PEST to continue or you can remove it completely and start the estimation process over again.



In the WinPEST dialogue the model can be stopped at any time using the stop icon. The pause icon does not stop execution but simply suspends it until you click on the play icon again. If you stop PEST, then by default it restarts the execution from scratch. However, you can change the restart options by selecting **[Options]** from the top menu and then **[Run/Restart]**.

This will bring up the following dialogue where you can select to restart PEST from either the last iteration or the last Jacobian matrix calculation.



If you choose to re-start the process using the last Jacobian matrix, PEST will move straight into calculation of the parameter upgrade vector and the testing of different Marquardt lambdas, based on the most recent completed Jacobian matrix.

A

Appendix A, 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 Appendix describes these file types in detail.

Template files, instruction files and control 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 utilities supplied in the WinPEST interface.

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".

PEST Template Files

Whenever PEST runs MODFLOW or MT3D, it must first write certain parameter values to the model input files. PEST provides the parameter values which it wants the model to use for a particular run. The only way the model can access these values is to read them from its input files. For example, 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 (i.e. text) input files. If a model requires a binary input file, you must write a program which translates data written to an ASCII file to binary form. The translator program, and then the model, can be run in sequence by placing them in a batch file which PEST calls 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. We suggest that template files have the extension ".TPL" to distinguish them from other types of file.

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 template file is replaced by a sequence of characters which identify the space as belonging to that parameter.

Aquifer properties that may need adjustment during model calibration include horizontal hydraulic conductivity, inter-layer conductance, storage coefficient, streambed or drain conductance and, in the case of transport models, dispersivity, the parameters defining adsorption isotherms, and solute decay constants. In other cases it may be necessary to adjust aquifer inputs such as recharge rates and concentrations, or the parameters governing evapotranspiration. These values are supplied to MODFLO and MT3D either through two-dimensional data arrays (for example hydraulic conductivity and dispersivity), or as cell-by-cell listings (for example drain and riverbed conductance). All of these parameters are distributed, i.e. a value is required for many (or all) model cells. As it is both practically infeasible and mathematically impossible to estimate a parameter value for each model cell using observations made at a discrete number of boreholes, these distributed parameters must be "regularised" in some way. The easiest way to do this is to assume that any such parameter type is "piecewise constant", i.e. that it takes on a single value in each of a number of discrete model sub-areas within the overall model domain.

PEST interfaces with a model through the models own ASCII input and output files. Each time PEST runs a model it first writes user-specified model input files using the parameter values which it wishes the model to use on that particular run. It knows where to write parameter values to input files through the use of model input file templates. For PEST to adjust a distributed parameter supplied to MODFLOW or MT3D through a two-dimensional array or cell-by-cell listing, a template must be constructed for the file which holds the array or listing. This is usually done by modifying a model input file, replacing parameter values with "parameter spaces" (comprising a parameter name enclosed by appropriate delimiters). Each parameter space denotes a contiguous set of characters on the model input file as belonging to a particular parameter. It also informs PEST of the number of digits which it may use to write the number representing the parameter.

Table 4: Template example for a two-dimensional array comprised of four different numbers

1.2345	1.2345	1.2345	1.2345	1.2345	6.7543	6.7543	6.7543
1.2345	1.2345	1.2345	1.2345	1.2345	6.7543	6.7543	6.7543
1.2345	1.2345	1.2345	9.6521	9.6521	6.7543	6.7543	6.7543
1.2345	1.2345	1.2345	9.6521	9.6521	9.6521	6.7543	6.7543
1.2345	1.2345	1.2345	9.6521	9.6521	9.6521	9.6521	6.7543
8.4352	1.2345	1.2345	9.6521	9.6521	9.6521	9.6521	9.6521
8.4352	8.4352	1.2345	9.6521	9.6521	9.6521	9.6521	9.6521
8.4352	8.4352	8.4352	9.6521	9.6521	9.6521	9.6521	9.6521
8.4352	8.4352	8.4352	8.4352	9.6521	9.6521	9.6521	9.6521

# par1 # # par1 # # par1 # # par1 # # par1 # # par2 # # par2 # # par2 #							
# par1 # # par1 # # par1 # # par1 # # par1 # # par2 # # par2 # # par2 #							
# par1 # # par1 # # par1 # # par3 # # par3 # # par2 # # par2 # # par2 #							
# par1 # # par1 # # par1 # # par3 # # par3 # # par3 # # par2 # # par2 #							
# par1 # # par1 # # par1 # # par3 # # par3 # # par3 # # par3 # # par2 #							
# par4 # # par1 # # par1 # # par3 # # par3 # # par3 # # par3 # # par3 #							
# par4 # # par4 # # par1 # # par3 # # par3 # # par3 # # par3 # # par3 #							
# par4 # # par4 # # par4 # # par3 # # par3 # # par3 # # par3 # # par3 #							
# par4 # # par4 # # par4 # # par4 # # par3 # # par3 # # par3 # # par3 #							

For a spatially distributed parameter occupying a two-dimensional array the model domain must be subdivided into a handful of zones where the parameter is constant. If each number in the array is replaced by an appropriate parameter space, the array of numbers as represented in the model input file becomes an array of parameter spaces. Each zone of parameter constancy within the array is then identified as having the same parameter name.

The first part of Table 4 illustrates a two-dimensional array of numbers subdivided into four zones of equal value. The second part of T able4 shows part of a template file constructed from it. Before PEST runs the model, it replaces the parameter spaces found in the template file by the current values pertaining to those parameters, thus building an array consisting of four separate numbers and defining four separate zones of parameter constancy

For parameters supplied to MODFLOW or MT3D on a cell-by-cell basis the cells can be divided into zones of similar value in the same way. For example, Table 5 shows part of a MODFLOW .DRN file for the Drain Package.

Table 5: Template example for part of the input to MODFLOW's DRN package.

```
 1     19     43 2.000E+01 3.000E+00
 1     20     43 2.000E+01 3.000E+00
 1     21     43 2.000E+01 3.000E+00
 1     22     44 2.000E+01 3.000E+00
 1     23     45 2.000E+01 3.000E+00
 1     24     46 2.000E+01 5.000E+00
 1     25     46 2.000E+01 5.000E+00
 1     26     46 2.000E+01 5.000E+00
 1     27     46 2.000E+01 5.000E+00
 1     28     45 2.000E+01 5.000E+00
 1     29     44 2.000E+01 5.000E+00
 1     30     43 2.000E+01 5.000E+00
 1     31     43 2.000E+01 5.000E+00
```

```
 1     19     43 2.000E+01 # con1  #
 1     20     43 2.000E+01 # con1  #
 1     21     43 2.000E+01 # con1  #
 1     22     44 2.000E+01 # con1  #
 1     23     45 2.000E+01 # con1  #
 1     24     46 2.000E+01 # con2  #
 1     25     46 2.000E+01 # con2  #
 1     26     46 2.000E+01 # con2  #
 1     27     46 2.000E+01 # con2  #
 1     28     45 2.000E+01 # con2  #
 1     29     44 2.000E+01 # con2  #
 1     30     43 2.000E+01 # con2  #
 1     31     43 2.000E+01 # con2  #
```

The drain has been subdivided into two zones in each of which the conductance is assumed uniform. (Note that in this example, the parameterization would probably benefit by tying all of the conductances to one conductance.

Visual MODFLOW's Template Files

Visual MODFLOW takes care of creating template files for the parameters that you select in the PEST Control dialogue. In this dialogue, you can currently select spatially variable anisotropic conductivities, storage parameters and recharge. The parameters that you select here are Visual MODFLOW parameters - not MODFLOW parameters. This means that you can select vertical hydraulic conductivity whereas in MODFLO this term is lumped into the vertical conductance variable.

Visual MODFLOW builds the MODFLOW input files before each run by using a combination of PERL source files (.SRC files) and template files that are written in C. PEST substitutes the current parameter value into the template file, which then creates the MODFLOW input file in the format outlined by the .SRC files. Tab le6 shows a typical Visual MODFLOW .TPL file that is set to optimise Kx for zones 1,2 and 3.

Table 6: Example Visual MODFLOW Template file

```
ptf #

sub adjust_g_format
{
   my $s = shift;
   $s =~ s/(\-?)(\d+)(\.?)(\d*)e(\+)(\d)(\d)(\@)/ $1$2$3$4e$6$7$8$9/g;
   $s =~ s/(\-?)(\d+)(\.?)(\d*)e(0)(\d)(\d)(\@)/ $1$2$3$4e$6$7$8/g;
   $s =~ s/(\-?)(\d+)(\.?)(\d*)e(0)(\d)(\@)/ $1$2$3$4e$6$7/g;
   $s =~ s/(\-?)(\d+)(\.?)(\d*)e(\-?)(0)(\d)(\d)(\@)/ $1$2$3$4e$5$7$8$9/g;
   $s =~ s/(\-?)(\d+)(\.?)(\d*)e(\-?)(0)(\d)(\@)/ $1$2$3$4e$5$7$8/g;
   $s =~ s/( )( *)(-?)(\d+)(\@)/$2$3$4\.$5/g;
   $s =~ s/\@//g;
   return $s;
}

sub Process
{
  my $source = shift;
  my $target = shift;
  open inp, "<$source";
  open out, ">$target";
  while(<inp>)
  {
    print out adjust_g_format eval $_;
  }
  close inp;
  close out;
}
$Kx__1 = #              Kx__1#;
$Kx__2 = #              Kx__2#;
$Kx__3 = #              Kx__3#;
undef %Source_Files_To_Process;
$Source_Files_To_Process{'D:\VMODNT\DEMP.BCF'} = 'D:\VMODNT\DEMP.BCF.SRC';
$Source_Files_To_Process{'D:\VMODNT\DEMP.WEL} = 'D:\VMODNT\DEMP.WEL.SRC';
foreach $target ( keys %Source_Files_To_Process )
{
  $source = $Source_Files_To_Process{$target};
  Process $source, $target;
}
```

Working Directly with MODFLOW/MT3D Files

We are expecting to continue to make additional Visual MODFLOW parameters available in the PEST Control dialogue. In this version, however, if you want to optimise a parameter that is not included in the list of available parameters, you will need to construct your own template files.

To prepare a template for a model input file you should first prepare the model input using Visual MODFLOW and translate the model without running it. Then construct the template file by first copying, and then modifying, the model input file containing the parameter to be optimised, replacing each zone value in the pertinent array or cell-by-cell listing by a corresponding parameter space. In this way, you will create a file that contains parameter spaces for the parameters that you want to optimise.

For example, suppose that you wish to optimise the amount of evapotranspiration and you have created the .EVT file by assigning the appropriate zones in Visual MODFLOW and translating the file. To prepare a template file named EVP.TPL from *projectname.evt*, copy *projectname.evt* to EVT.TPL, add a template file header (e.g. "ptf &", "&" being the parameter delimiter) and use the "search and replace" facilities of a text editor to replace each occurrence of each zone-defining value in the maximu ET array by an appropriate parameter space to produce an array like the one shown in Table 4.

Unfortunately this process is not always as simple as it sounds. MODFLOW and MT3D input files can be very large and your text editor may not be able to read them (We use MultiEdit for Windows(r) by American Cybernetics; www.amcyber.com). Also, you must be carefule to ensure that the "search and replace" does not make changes beyond the target area of the MODFLOW/MT3D input file. Furthermore, if you want to make changes to the zonation of a parameter then you will have to re-translate the files and you must re-create the template file. Also you should be careful not to try optimising parameters that lie completely within an inactive zone. Your template files can be checked using the WinPEST file checking routines.

Working with files created by Visual MODFLOW

Sometimes the parameter values that you supply to a model preprocessor are not actually reproduced in the MODFLOW/MT3D input files written by Visual MODFLOW. This may cause problems when easily-identified numbers are supplied to a particular parameter with the aim of replacing those numbers later by parameter spaces. In Visual MODFLOW, this can be caused by internal unit conversion or because MODFLOW does not use the parameter as it is supplied to Visual MODFLOW. For example, pumping rates are specified in the MODFLOW files in units consistent with other length and time parameters.

MODFLOW defines four different types of model layers using via the LAYCON parameter. If LAYCON for a layer is 0 or 2, MODFLOW expects a transmissivity array for that layer. However if the LAYCON element is 1 or 3, MODFLOW expects a hydraulic conductivity array. Since, Visual MODFLOW uses only hydraulic conductivity, if a layer is of type 0 or 2 the hydraulic conductivity is multiplied by the layer thickness to obtain transmissivity which is then translated to the MODFLO input file. Therefore, user-supplied cell hydraulic conductivity values are not replicated in the MODFLOW .BCF file. Furthermore, if the layer thickness is irregular, the transmissivity array will not be piecewise constant even if the hydraulic conductivity was entered in zones of constant value. In such a case, it is better to supply a uniform

aquifer thickness to the preprocessor even if the aquifer thickness is, in fact, variable. For layers of type 0 or 2 this inaccuracy will not degrade model results as it is the transmissivity, and not the hydraulic conductivity and layer thickness individually, which determines the flow regime within the aquifer.

Similar considerations apply to the VCONT array required by MODFLOW for multi-layered models, which represents the vertical hydraulic conductivity divided by thickness.

Note that only arrays and cell-by-cell parameters that are to be optimised need to be considered when constructing a template file. Arrays and values which will not be used in the parameter optimisation should be left unaltered in the template file. Similarly, if parameter optimisation is sought for only part of a model domain, then only part of the array needs to have its elements replaced by parameter spaces when constructing the template file.

Multi-Array Parameters and Tied Parameters

There is no reason why the occurrence of a particular parameter should be restricted to a single array. For example, if a single, vertically homogeneous aquifer is represented by a number of model layers, the arrays representing the hydraulic conductivity of each model layer will be identical. In such a case, the parameter space arrays on the corresponding template file will also be identical, each such array containing the same parameters in the same disposition. The VCONT arrays for model layers within the same aquifer will also be identical from layer to layer (if the layer thicknesses are the same). The pertinent parameter values may be estimated separately from hydraulic conductivity, or they may be tied to the latter, if the relationship between vertical and horizontal hydraulic conductivity is known. In the latter case, only the horizontal hydraulic conductivity needs to be estimated, the estimates for VCONT tracking the horizontal values as the optimisation process progresses.

Similarly, for parameter types such as drain conductance, it may be opportune to define a small number of conductances over the model domain, assuming that the hydraulic conductivity of the drain material is uniform, the conductance in a particular cell will be proportional to the length of drain in that cell. Thus, only one conductance parameter needs to be optimised, the others being tied to it in proportion to the respective length category represented by each parameter.

Fixed and Transformed Parameters

A parameter must be fixed if it lies wholly within the inactive part of the model grid and, hence, has no effect on model outcomes. A parameter should also be fixed if its effect on model outcomes at all observation points is particularly weak. Likewise, if a group of highly correlated parameters is identified, then at least one member of this group may need to be fixed to stabilise the optimisation and to make the estimation of the remaining members of the group more efficient.

The logarithmic transformation of certain parameter types, such as conductivity may have a dramatic effect on optimisation efficiency, as will appropriate upper and lower bounds for adjustable parameters.

Template File Syntax and Commands

The Parameter Delimiter

As Table 6 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.

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 eight 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 eight 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.

Setting the Parameter Space Width

In general, the wider the parameter space (up to a certain limit - see below), the better PEST likes it, since numbers can be represented with greater precision in wider spaces than they can be in narrower spaces. However, unlike model-generated observations where maximum precision is crucial to obtaining useable derivatives, PEST can adjust

to limited precision for parameters in input files. Enough precision needs to be available for the parameter value to be distinguished between iterations after it has been 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 used, extra precision is not critical. Nevertheless, it is good practice to use as much precision as the model is capable of reading the parameters with, so that they can be provided to the model with the same precision with which PEST calculates them.

In MODFLOW and MT3D, the FORTRAN file formats are found in their respective Reference Manuals. For example, the following FORTRAN code directs a program to read five real numbers. The first three are read using a format specifier, whereas the last two are read in list-directed fashion.

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

The relevant part of the input file may be

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

Notice how no whitespace or comma is needed between numbers which are read using a field specifier. The format statement labelled "100" 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 for the program to know 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 may then be

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

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. If the parameter space for any of these parameters was greater than 10 characters in width, then PEST, when it replaced the parameter space by the current parameter value, it would create a model input file which would be incorrectly read by the model. You could also designed parameter spaces less than 10 characters wide if you wished, as long as there were enough spaces between each parameter space so that the value falls within the field expected by the

model. However, there is no advantage in using less than the full number of characters allowed by the model.

In the above example, parameters D and E are treated very differently to parameters A, B and C. In this case, the model simply expects two numbers in succession. If the spaces for parameters D and E are replaced by two numbers (each will be 13 characters long) the model's requirement for two numbers separated by whitespace or a comma is satisfied, as is PEST's preference for maximum precision.

Comparing the two lines above, it is obvious that the spaces for parameters D and E in 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. In most cases of template file construction, a model input file will be used as the starting point. In such a file, numbers read as list-directed input will often be written with trailing zeros omitted. In constructing the template file you should recognise which numbers are read using list-directed input and expand the parameter space (to the right) accordingly beyond the original number, making sure to leave whitespace or a comma between successive spaces, or between a parameter space and a neighbouring character or number

Similarly, numbers read through field-specifying format statements may not occupy the full field width in a model input file from which a template file is being constructed (e.g. variable A in the example above). 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.

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 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 found in the PEST control file, but only PRECIS can be modified in the Visual MODFLOW PEST Control dialogue. PRECIS is a character variable which must be either "single" or "double". It determines whether single or double precision is used to write 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 list directed input, or using a field width specifier such as "(F6.0)" or "(E8.0)", the decimal point is not necessary. However, in other cases the format specifier will insert its own decimal point (e.g. for specifiers such as "(F6.2)"), or enforce power-of-10 scaling (e.g. for specifiers such as "(E8.2)") if a decimal point is absent from an input number. Hence, if you are unsure what to do, assign the string "point" to the control variable DPOINT. This will ensure that all numbers written to model input files will include a decimal point, thus overriding point-location or scaling conventions implicit in some FORTRAN format specifiers. Visual MODFLOW uses the a value of “point” for the DPOINT variable.

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.

A template file may contain multiple occurances of the same parameter. If the parameter spaces for that parameter are defined differently, PEST will write the parameter value to all of its parameter 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. This way a consistent parameter value is written to all parameter spaces for that parameter.

The Same Parameter in Different Files

Multiple incidences of the same parameter are not restricted to one file. PEST passes no judgement on the occurrence of parameters within template files or across template files. However, it does require that each parameter cited in the PEST control file occur at least once in 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.

PEST Instruction Files for Output

Of the voluminous amounts of information that MODFLOW and MT3D writes, PEST is interested in only a few numbers. That is, those output values (“observations” or “model-generated observations”) for which corresponding field or laboratory data (“measurements”) are available and for which the discrepancy between model output and measured values is part of the objective function.

For every model output file containing observations, you must provide an instruction file (*.INS) containing the directions which PEST must follow to read the file.

Precision in Model Output Files

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 non-linear parameter estimation, on 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.

How PEST Reads Model Output Files

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 binary files.

Unfortunately, observations cannot be read from model output files using the template concept, since neither MODFLOW nor MT3D cannot be relied upon to produce an output file of identical structure during each model run. So instead of using an output file template, you must provide PEST with a list of instructions on how to find observations in the output files (see Table 7).

Basically, PEST finds observations in a model output file in the same way that you would. You run your eye down the file looking for something you recognise - a "marker". If this marker is properly selected, observations can usually be linked to it. For example, if you are looking for the output after 100 days, you may look for

```
TIME = 100 DAYS
```

A particular outcome for which you have a corresponding field measurement may then be found, for example, between character positions 23 and 30 on the 4th line following

Table 7: Example output file and corresponding PEST instruction file.

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

```
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
```

the marker. For output files, a marker may be unnecessary as the default initial marker is 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.

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.

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 eight characters or less in length. These eight 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 Table 7, observations are named "ar1", "ar2" etc. These same observation names must also be cited in the PEST control file where measurement values and weights are provided.

There is one observation name, however, to which these rules do not apply, that is 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.

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, 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 inTable 8 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.

Table 8: Example for a more complex output and instruction file

```
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
```

```
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
```

Line Advance

The syntax for the line advance item is "ln" where n is the number of lines to advance. 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 n'th 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 Table 7 model-calculated values 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

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 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.

Table 9 shows an extract from a model output file and 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.

Table 9: Example instruction file with secondary markers

DISTANCE = 20.0: CATION CONCENTRATIONS:- Na: 3.49868E-2 Mg: 5.987638E-2 K: 9.987362E-3
pif ~ ~DISTANCE = 20.0~ l1 ~K:~ !kc!

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 Table 9. 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.

Table 10: Instruction file with qualified secondary markers

```
.
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
.
```

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

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.

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 in 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 n1 and n2 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".

Table 11 shows how the numbers listed in the third solution vector of Table 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).

Table 11: Alternative instruction set for output in Table 8

```
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
```

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,
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

Table 8 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, that is, 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

Table 10 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 and instruction file shown in Table 12. 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 "adjusted" may be required in the calculation of any such population. Hence, we must find our way to the number using a method such as that illustrated in Tabl e12.

Table 12: Output file that cannot read as fixed or semi-fixed

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

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

A primary marker is used to move the PEST cursor to the first of the lines shown in Table 12. 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 can use neither a secondary marker nor a tab. Fortunately, whitespace comes to the rescue, with the following instruction line taking PEST to the fourth number:

l10 w w w !obs1!

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

l10 !dum! w w w !obs1!

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

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

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

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

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

SOIL WATER CONTENT (NO CORRECTION)=21.345634%

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

l5 *=* !sws! *%*

Notice how a secondary marker (i.e. *%*) is referenced even though it occurs after the observation we wish to read. If this marker were not present, a run-time error would occur when PEST tries to read the soil water content because it would define the observation string to include the "%" character and, naturally, would be unable to read a number from a string which includes non-numeric characters. However, by including the "%" character as a secondary marker after the number representing the observation 'sws', PEST 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.

Creating and Checking an Instruction File

The Instruction files can be created in using any text editor. However, caution must be exercised in building an instruction set to read a model output file, especially if navigational instructions such as markers, whitespace, tabs and dummy observations are used. PEST will always follow your instructions to the letter, but it may not read the number you intend 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 the WinPEST environment, you can check your instruction file using the built in checking routines.

The PEST Control File

The PEST control file contains all of the parameter and control values for the PEST run and must have the extension .PST. Many of the data items in the PEST control file are used to "tune" PEST's operation to the current project. Such items include parameter change limits, parameter transformation types, termination criteria etc.

The PEST control file is automaticaly built by Visual MODFLOW, but it can be easily edited using a text editor, if necessary. However, every time your project is translated, the .PST file will be re-created.

The PEST control file consists of integer, real and character variables. Its construction details are shown in Tabl e13, where variables are referenced by name. Chapter 4 contains detailed descriptions of the parameters and how they are entered in Visual MODFLOW. In Table 14 is an example PEST Control File, which was used by PEST to create the PEST Run Record found in Appendix B.

A PEST control file must begin with the letters "pcf" for "**P**EST **c**ontrol **f**ile". Scattered through the file are a number of section headers. These headers always follow the same format, i.e. an asterisk followed by a space followed by text. When preparing a PEST control file, these headers must be written exactly as shown.

Table 13: PEST control file structure.

* control data
RSTFLE
NPAR NOBS NPARGP NPRIOR NOBSGP
NTPLFLE NINSFLE **PRECIS** DPOINT
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
(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
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
(one such line for each of the NPRIOR articles of prior information)

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 done with either the "d" or "e" symbol. Note, however, that all real numbers are stored internally by PEST as double precision

numbers. Descriptions for the parameters in bold in Table13 c an be found in Chapter 4. The following is a brief description of the parameters not found in Chapter 4.

NPAR - total number of parameters

NOBS - total number of observations

NPARGP - number of parameter groups

NPRIOR - number of articles of prior information

NTPLFLE - number of model input files that contain parameters. There must be one template file for each model input file.

NINSFLE - number of instruction files. There must be one instruction file for each output file containing model-generated observations.

DPOINT - character variable, “point” or “nopoint” which tells PEST whether to include the decimal point with whole numbers.

PARTIED - the name of the parent parameter for tied parameters

OBGNME - observation group name

OBSNME - observation nam

OBSVAL - observation value

WEIGH - observation wieght

TEMPFLE - template file name for model input

INFLE - model input file to which the template file is matched

INSFLE - instruction file for model output

OUTFLE - model output file to which the instruction file is matched

PILBL - prior information label

PIFAC - prior information factor

PIVAL - prior information value

WEIGH - weight associated with article of prior information

Table 14: Example PEST control file.

```
pcf
* control data
restart
5 19 2 2 3
1 1 single point
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
ro2  log   factor  5.0   .1  10     ro   1.0 0.0
ro3  tied  factor  0.5   .1  10     ro   1.0 0.0
h1   none  factor  2.0   .05 100    h    1.0 0.0
h2   log   factor  5.0   .05 100    h    1.0 0.0
ro3  ro2
* observation groups
group_1
group_2
group_3
* 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
pi2  1.0 * log(ro2) + 1.0 * log(h2) = 2.6026 2.0
```

B

Appendix B, A PEST Run Record

In this Appendix, an example PEST Run Record is illustrated and described in detail. The Run Record is generated from the PEST Control file in Appendix A.

Note that this example does not demonstrate a very good fit between measurements and model outcomes calculated on the basis of the optimized 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 subsections.

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 (see Appendix A). Note that the letters "na" stand for "not applicable". "na" is used a number of times to indicate that a particular PEST input variable has no effect on the optimization 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. 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. For consistency, PEST's internal representation of parameter bounds is adjusted in the same way.

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 optimization iteration. After calculating the

Jacobian matrix PEST attempts objective function improvement using one or more Marquardt lambda's. 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 optimization iteration.

During the first iteration in this example, PEST tests two Marquardt lambda's; because the second lambda results in an objective function fall of less than 0.03 (i.e. PHIREDLAM) relative to the first one tested, PEST does not test any further lambda's. Instead it progresses to the next optimization 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 optimization iteration. Note that the only occasion on which the "previous parameter values" recorded at the end of an optimization iteration do not correspond with those determined during the previous optimization 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 optimization iteration the parameter set resulting in the lowest objective function value achieved so far.

At the end of each optimization iteration PEST records either two or three (depending on the input settings) very important pieces of information. In this example it is two. These are the maximum factor parameter change and the maximum relative parameter change. As was discussed in Chapter 2, each adjustable parameter must be designated as either factor-limited or relative-limited. In this example, all adjustable parameters are factor-limited with a factor limit of 3.0. A suitable setting for the factor and relative change limits (i.e. FACPARMAX and RELPARMAX) may be crucial in achieving optimization stability. Note that, along with the value of the maximum factor or parameter change encountered during the optimization 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 held temporarily fixed should trouble be encountered in the optimization process.

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 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. Had some of the parameters in this example been relative-limited, this part of the run record would have been slightly different. In this case, 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.

This PEST run record shows that in iteration 2, one of the parameters, "h2", incurs the maximum allowed factor change, thus limiting the magnitude of the parameter upgrade vector. In optimization 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 optimization 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 Chapter 2. Note that at the beginning of optimization iteration 3, parameter "ro2" is released again in case, with the upgrading of the other adjustable parameters during the previous optimization iteration, it wants to move back into the internal part of its domain.

In the third optimization 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 optimization iteration three lambda's 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 5.4, 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 optimization 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 (i.e. RLAMFAC) than the most successful lambda of the previous iteration.

At the end of optimization iteration 6 PEST calculates that the relative reduction in the objective function from that achieved in iteration 5 is less that 0.1; i.e. it is less than the user-supplied value for the PEST control variable PHIREDSWH. Hence, as the input variable FORCEN for at least one parameter group (both groups in the present example) is set to "switch", PEST records the fact that it will be using central differences to calculate derivatives with respect to the members of those groups from now on. Note that in this example, 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 optimization process moving again.

The optimization process of this example is terminated at the end of optimization iteration 7, after the lowest 3 (i.e. NPHISTP) objective function values are within a relative distance of 0.01 (i.e. PHIREDSTP) of each other.

Note that where PEST lists the current objective function value at the start of the optimization process and at the start of each optimization iteration, it also lists the contribution made to the objective function by each the observation groups and by all prior information. This is valuable information, for if a user notices that one particular group, or the prior information equations, are either dominating the objective function or are not “seen” because something else was dominating, the observation or prior information weights could be adjusted and the optimization process started again.

Optimized 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 optimized 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 (e.g. 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 optimization 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 much more exaggerated in the domain of non-logarithmically-transformed numbers. This is readily apparent in this example.
- 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 this example, the upper confidence limits for both "ro2" and "h2" lie well above the allowed 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 optimized 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 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 this example, 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, which 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 optimization run a listing of the optimized parameter values can also be found in the PEST parameter value file, *projectname*.PAR.

Observations, Prior Information and Residuals

After it has written the optimized 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 optimized parameter set. The differences between the two (i.e. the residuals) are also listed, together with the user-supplied set of observation weights. Tabulated residuals and weighted residuals can also be found in file *projectname*.RES.

Following the observations, the user-supplied and model-optimized 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.

The Covariance Matrix

If the PEST input variable ICOV is set to 1, the parameter covariance matrix is written to the run record file. 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 optimized 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.) Hence in this example, the row (and column) order is "ro2", "h1", "h2".

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 insofar 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 this example 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; i.e. 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 a new line. Note, however, that every new row of the covariance matrix begins on a new line.

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 this example, the logs of parameters "ro2" and "h2" are highly correlated, 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.

The Normalized Eigenvector Matrix and the Eigenvalues

PEST calculates the normalized eigenvectors of the covariance matrix and their respective eigenvalues. The eigenvector matrix is composed of as many columns as there are adjustable parameters, each column containing a normalized 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 each eigenvector is dominated by a single element, then each adjustable parameter is well resolved by the parameter estimation process. However, where the dominance of eigenvectors is shared by a number of elements, parameters may not be well resolved; the higher the eigenvalues of those eigenvectors for which dominance is shared by more than one element, the less susceptible are the respective individual parameters to estimation.

In this example, the eigenvector of highest eigenvalue is dominated by two parameters, these being log("ro2") and log("h2"). Thus, the parameter estimation process individually, poorly discerns these parameters, as the width of their confidence intervals demonstrates. However, the second highest eigenvector is dominated by the single parameter "h1" which, in comparison with the other parameters, is well resolved.

The PEST Run Record for the Control file in Appendix

```
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.tp1
  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 name	Factor		Parameter		Prior information	Weight
pi1	1.00000	*	h1	=	2.00000	3.000
pi2	1.00000	*	log[ro2]	+		
	1.00000	*	log[h2]	=	2.60260	2.000

Observations:-

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 prior information = 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 prior information : 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 prior information         : 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 prior information         : 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 prior information          : 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 prior information          : 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 prior information         : 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 prior information : 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 value	95% percent confidence limits 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 value	Calculated value	Residual	Weight	Group
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 information	Provided value	Calculated value	Residual	Weight
pi1	2.00000	0.261177	1.73882	3.000
pi2	2.60260	2.62532	-2.271874E-02	2.000

Sum of squared weighted residuals (ie phi) = 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 prior information = 27.24

Covariance Matrix ----->

0.3136	4.8700E-03	-0.4563
4.8700E-03	0.3618	1.3340E-02

-0.4563 1.3340E-02 0.8660

Correlation Coefficient Matrix ----->

1.000 1.4457E-02 -0.8756

1.4457E-02 1.000 2.3832E-02

-0.8756 2.3832E-02 1.000

Normalized eigenvectors of covariance matrix ----->

-0.8704 -3.6691E-02 -0.4909

3.5287E-02 -0.9993 1.2121E-02

-0.4910 -6.7718E-03 0.8711

Eigenvalues ----->

5.6045E-02 0.3621 1.123

Index

R

W