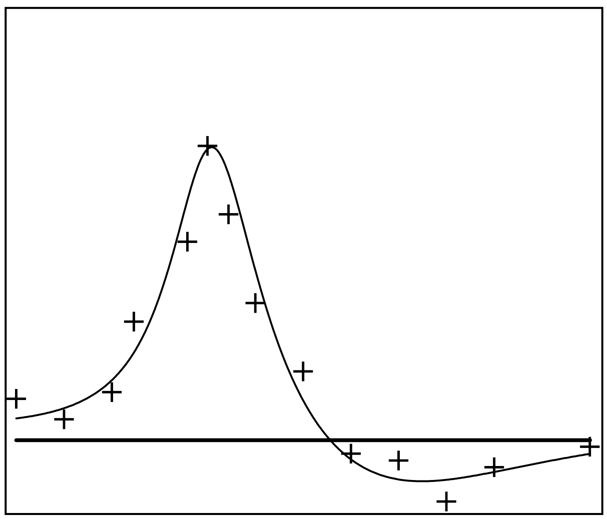
PEST_HP

PEST for Highly Parallelized Computing Environments



Watermark Numerical Computing

January, 2025

Table of Contents

1. Introduction

1.1 General

"HP" stands for "highly parallelized" or "high performance".

The HP suite is comprised of a growing number of programs, some of which are counterparts to programs of similar name in the PEST suite, and some of which do not have PEST suite counterparts. All of them read a PEST control file. All of them require that a model be run many times. All of them provide a non-intrusive interface with a model using template and instruction files.

This manual describes PEST_HP, as well as two programs that were written specifically to support the use of PEST_HP, namely PWHISP_HP and PCOST_HP. Other members of the HP suite are described in a separate manual.

All members of the HP suite undertake parallelized model runs using a similar protocol to that employed by BEOPEST. Their algorithms have been tuned for operation in parallel computing environments. Model runs can be undertaken on one or on many computers. However while no member of the HP suite has been programmed to undertake serial model runs, there is no reason why a single stream of model runs cannot be undertaken on a single machine. The algorithm employed by any member of the HP suite can readily adapt to whatever parallelization environment is available to it.

1.2 PEST_HP in Brief

PEST_HP is a heavily modified version of BEOPEST. Its inversion algorithm is similar to that of PEST (and hence BEOPEST). However parts of that algorithm have been altered to improve its performance when working in conjunction with big models with long run times and questionable numerical stability. Its inversion algorithm has also been tuned for increased efficiency in highly parallelized computing environments such as high performance clusters and the computing cloud.

There are some elements of PEST functionality which PEST_HP does not possess. In developing PEST_HP, the PEST source code has been renovated, and some of it removed, to make it easier to alter and debug. At the same time, those aspects of the PEST inversion algorithm that pertain to highly parameterized inversion and parallel run management have been enhanced and made considerably more efficient.

The following capabilities of PEST/BEOPEST have been eliminated in producing PEST_HP.

- PEST_HP cannot run in "predictive analysis" mode.
- It cannot read an external derivatives file.
- It cannot write a "PEST-to-model-message" file.
- It does not write a resolution data file.
- PEST_HP cannot be started using the "/p1" switch to activate conjunctive parallelisation of the first model run with those required for filling the first Jacobian matrix. However this capability has been replaced with functionality that achieves a similar outcome – see below.

- PEST_HP does not write a sequence of matrix files in which approximations to the posterior parameter covariance and related matrices are recorded during every iteration of the inversion process. (Actually this type of file is available through PEST only if the inversion process is over-determined and unregularized.)
- Manual and automatic user intervention are not supported by PEST_HP.
- PEST_HP does not record the AIC and similar information-theoretic statistics on its run record file. (These are meaningless in contexts of highly parameterized inversion.)
- On completion of an inversion process, PEST_HP does not undertake a final model run using optimised parameters.

Most of the above PEST capabilities are not widely used. Hence contexts in which PEST_HP can work are generally the same as those in which PEST (and BEOPEST) can work. PEST_HP reads a normal PEST input dataset. If a PEST control file instructs PEST_HP to undertake processing which it does not support, it ceases execution with an appropriate error message.

The following features are unique to PEST_HP. Collectively, they generally result in improved numerical performance of PEST_HP over that of BEOPEST when conducting highly parameterized inversion in a highly parallelized computing environment. They also result in improved reporting of that performance. All of these new features are discussed in this manual.

- During each iteration of the inversion process, PEST_HP calculates a series of Marquardt lambda values on which to base parameter upgrade calculations. The number of selected lambdas depends on the number of cores available for testing parameter upgrades. Each lambda defines a direction in parameter space; lambdas are selected using an enhanced version of the algorithm employed by the traditional PEST. PEST_HP then chooses a number of parameter sets along each of these directions as parameter upgrade candidates. These parameter sets are then distributed to networked cores for parallelized testing. This process keeps otherwise idle cores busy at the same time as it maximizes the chances of finding an improved parameter set, even in difficult numerical circumstances where the integrity of finite-difference-based derivatives may be compromised by model convergence difficulties.
- Optionally, a second round of parameter upgrade calculations and parallelized testing can be undertaken using a Jacobian matrix that is improved through Broyden updating.
- The efficiency of the BEOPEST parallel run manager has been improved. If requested, model runs can be abandoned and/or terminated during the parameter upgrade testing procedure if they take too long. Thus if testing of a problematic parameter set promulgates model solver convergence difficulties, the inversion process can continue without having to wait for completion of the afflicted model run.
- In order to reduce idle processor time, and hence to raise parallelization efficiency, the initial model run of the inversion process can be undertaken in a serial computing environment prior to using PEST_HP for implementation of the inversion process in a parallelized environment. This pre-inversion run also internally orders observations recorded in the PEST control file to match the reading sequence encountered in model output files. Where observation numbers are high, the efficiency of reading model output files can be greatly improved through this process.
- The efficiency of imposition of parameter bounds in highly parameterized contexts has been improved.

- PEST's sensitivity reuse functionality has been improved to better accommodate a parallel computing environment.
- PEST_HP delays the onset of higher order finite-difference derivatives calculation if it detects that imposition of parameter change limits have hindered improvements to the objective function.
- PEST_HP provides a continuous record of parallel computing efficiency; it also provides a record of parameter sets that instigate model run failure.
- In undertaking parameter estimation, PEST_HP can optionally write a "run results file" which records parameters and model outputs pertaining to all non-Jacobian model runs. In future versions of PEST_HP, the contents of this file may support construction of proxy models.
- The running of PEST_HP can be dedicated solely to production of a run results file based on parameter sets provided in a series of parameter value files. This can facilitate uncertainty analysis, sensitivity analysis and proxy model construction.
- PEST_HP can be instructed to cease execution after a user-specified time has elapsed, regardless of whether the inversion process is complete or not.
- Where multiple commands are used to run a model, or where PEST's observation re-referencing functionality is activated, user-specified numbers written to model output files can instruct PEST_HP to award zero sensitivities of respective model outputs to a parameter that is being varied incrementally for finite-difference derivatives calculation.
- Model-generated files can be distributed to all computing nodes (possibly after local processing) prior to the commencement of each iteration of the inversion process; in some circumstances this can lead to considerably enhanced model execution speeds.
- A PEST_HP control file can cite new parameter types, namely "secondary parameters". Use of secondary parameters can replace the PAR2PAR utility as a means of transforming parameters that are estimated through the inversion process to those that are actually used by the model.
- PEST_HP provides a novel means of obtaining a Jacobian matrix based on randomized finite differences. This methodology is still in its experimental stages.
- User-specified elements of a Jacobian matrix can be strategically set to zero. This can raise the integrity of a Jacobian matrix that has been filled through random parameter differencing.
- Where a Jacobian matrix is known to possess a blocky structure, PEST_HP can increment multiple parameters simultaneously. This, together with its Jacobian blanking functionality, can considerably reduce the model run burden of filling a Jacobian matrix in some inversion circumstances.
- A new "quick regularisation" option is available.

Meanwhile, the following (and many other) advanced features of PEST/BEOPEST are retained in PEST_HP.

- SVD-assisted inversion can be employed for estimation of parameters, and for production of calibration-constrained random parameter sets using the null space Monte Carlo methodology.
- One objective function can be traded off against another by running PEST_HP in "pareto" mode.

- Inverse problem solution strategies based on Gaussian elimination, singular value decomposition and LSQR are all supported.
- Multiple options are provided for calculation of regularisation weights, and for determination of comparative inter-regularisation-group weighting when applying Tikhonov regularisation.
- Different command lines can be used to run the model in calculating derivatives with respect to different parameters.
- Observation re-referencing supports use of one or a number of simplified models for calculation of derivatives, while retaining a complex model for testing parameter upgrades.
- Alterations made to members of the PEST suite (versions 15 and above) to promote greater inter-operability with members of the PEST++ suite have also been made to members of the HP suite. These include the use of comment lines in a PEST control file. See PEST documentation for details.

PEST_HP can be used interchangeably with PEST and BEOPEST. It requires no input data in addition to that required by the normal PEST; hence a PEST input dataset is also a PEST_HP input dataset. However a number of PEST_HP-specific control variables, and three sections, can optionally be added to a normal PEST control file to construct a PEST_HP control file. PEST and BEOPEST will cease execution with a pertinent error message if they encounter these new variables and sections. Conversely, PEST_HP will cease execution with an appropriate error message if a PEST input dataset requests functionality which is no longer available in PEST_HP.

Note that the binary restart files written by PEST/BEOPEST on the one hand, and by PEST_HP on the other hand, are not compatible with each other. Hence an inversion process which is commenced by one of these programs cannot be completed by the other.

1.3 Terminology

The "master/slave" terminology employed by BEOPEST has been replaced by new nomenclature. The "master" is now the "manager". A "slave" is now an "agent". All members of the HP suite (including PEST_HP itself) employ a common agent; this is named AGENT_HP. The running of PEST_HP and AGENT_HP are described below.

A manager has two roles. Firstly, it undertakes the set of numerical tasks, and implements the set of numerical algorithms, that define it. Secondly, it commissions model runs that are required by its algorithm. Each of these runs is undertaken under the supervision of an agent. Many agents can exist. The manager allocates each required model run to the fastest available agent, and monitors all agents for completion of model runs. It can accommodate late and overdue model runs and missing agents.

The software that actually manages model runs is not a discrete executable program. Rather it is a set of functions and subroutines that are called by all members of the HP suite. Hence while each member of the HP suite performs a different numerical task, all members of the HP-suite call the same run management functions and subroutines. Because of this, control variables featured in a PEST control file which govern model run management have the same roles for all members of the HP suite.

Each agent must operate from a separate folder. This folder must contain the PEST control file for the current problem, all template and instruction files cited in the PEST control file, as well as all files that are required to run the model. See documentation of BEOPEST for further details.

The version number of AGENT_HP must be the same as that of the HP suite manager program with which it interacts. If this is not the case, the manager will detect this inconsistency and cease execution with an appropriate error message.

1.4 Using PEST_HP

1.4.1 Starting PEST_HP

Use of PEST_HP is very similar to that of BEOPEST. To run the PEST_HP manager based on the contents of the PEST control file *case.pst*, start it using the command

```
pest_hp case /h :nnnn
```

where *nnnn* is a port number (for example 4004). Note the space character between the "/h" string and the following colon.

Unlike BEOPEST, PEST_HP does not operate as both a manager and an agent. Instead a general HP named AGENT_HP supervises each local model run. AGENT_HP can be used with all members of the HP suite. To run AGENT_HP, use the command

```
agent_hp case /h hostname:nnnn
```

where *hostname* is the name of the machine on which PEST_HP (or any other member of the HP suite) is running, and *nnnn* is the port number employed by PEST_HP (as provided on its command line). Alternatively use the IP address of the machine on which PEST_HP is running instead of *hostname*. (Only TCP/IP version 4 is supported at the time of writing.)

Execution of many agents can be initiated in the manner described above. As stated above, each parallel run agent must operate from a different folder. If desired, one of these folders can be the same as that employed by a HP suite manager (e.g. PEST_HP). Execution of an agent can be initiated either before or after that of PEST_HP.

If only a single agent is employed, and if PEST_HP and the agent are both operating from the same folder, then a user may wish to initiate execution of both PEST_HP and AGENT_HP using a single command. This command initiates execution of a batch script whose contents are shown in figure 1.1. The file containing this script must be prepared by you, the user. Suppose that you name this file *run_pest_hp.bat*. Then execution of PEST_HP and a single agent can be initiated using the command

```
run_pest_hp case
```

where *case.pst* is the name of the PEST control file which defines the current inversion problem.

```
echo agent_hp %1 /h %computername%:4004 > agent.bat
start agent.bat
pest_hp %1 /h :4004
```

Figure 1.1 A batch file that can initiate execution of both PEST_HP and a single agent.

In the above script *computername* is an environment variable that returns the name of a computer. The script depicted in figure 1.1 can be easily modified for initiation of multiple agents running in multiple folders.

1.4.2 Termination of PEST_HP

Operation of PEST_HP can be terminated in the same manner as that of PEST and BEOPEST. This can be done brutally by pressing the <Ctl-C> keys when focussed on the PEST_HP window. Alternatively, the PSTOP or PSTOPST command can be issued from another window open to PEST_HP's working folder.

A halted PEST_HP run can be resumed using the "/s" switch. This switch must be placed before the "/h" switch in the PEST_HP command line. Hence to recommence an interrupted PEST_HP run based on file *case.pst*, use the command

```
pest_hp case /s /h :nnnn
```

The "/s" switch is not required when recommencing execution of AGENT_HP agents.

1.4.3 Other PEST_HP Command Line Switches

The "/i" command line switch and the "/t" command line switch have the same roles for PEST_HP as they do for BEOPEST. The "/i" switch informs PEST_HP that it must read a Jacobian matrix file (i.e. a JCO file) to obtain the Jacobian matrix for use in the first iteration of the inversion process; if started using this switch, PEST_HP prompts for the name of the JCO file which it must read. The "/t" switch allows a user to associate a text string with a particular PEST_HP run; see PEST documentation for details.

A new switch, namely "/hpstart", has been introduced for the use of PEST_HP, as well as for that of PEST; it enables the latter to expedite the efficiency of an ensuing PEST_HP run. When employed on the PEST_HP command line, the "/hpstart" option instructs PEST_HP to read a "hp starter file" named *case.hp* (where the PEST control file is named *case.pst*). This file contains information previously recorded during a dedicated PEST run, also initiated using the "/hpstart" command line option, in which the model is run only once. The information contained in file *case.hp* saves PEST_HP from having to conduct an initial model run to calculate the initial objective function and initial model-calculated reference values associated with observations comprising the calibration dataset. Hence, immediately upon commencement of its execution, PEST_HP can initiate parallelization of model runs in order to fill the initial Jacobian matrix. At the same time, where observations number in the tens or hundreds of thousands, PEST_HP's reading of the model-generated counterparts of observations from model output files is accelerated, as observations will have been ordered internally to PEST_HP to match their occurrence in these files. This is further discussed later in this manual.

1.5 An Alternative Version of PEST_HP

Two versions of the PEST_HP executable program are provided. One is named *pest_hp.exe*. The other is named *pest_hp_mkl.exe*. The latter executable program is linked to the Intel Maths Kernel Library. At the same time, the programming associated with a number of numerically-intensive parts of the PEST_HP algorithm have been altered to take advantage of the efficiencies offered by this library. Where the number of parameters and/or observations is large, this can result in much faster execution of PEST_HP.

If you use *pest_hp_mkl.exe* then a DLL (i.e. a dynamic linked library) named *libiomp5md.dll* must be situated in the folder from which you run *pest_hp_mkl.exe*. Alternatively, it must reside in a folder that is accessible through your computer's PATH environment variable. Many users will already have this DLL resident on their machines. However, because some will not, it is supplied with PEST_HP.

1.6 Parallel Run Management File

As is discussed in PEST documentation, Parallel PEST (a version of PEST which uses so-called "message files" for communication between managers and agents) requires that the user construct a parallel run management file prior to initiating its execution. This file must possess the same filename base as that of the PEST control file, but its extension must be "*.rmf*". BEOPEST does not require a file of this type. However, if this file is present in the directory from which it is run, BEOPEST will read values for the optional PARLAM and RUN_SLOW_FAC control variables from this file. In contrast, PEST_HP does not read a parallel run management file at all. As will be discussed below, the PARLAM variable is redundant to parallel run management undertaken by PEST_HP; if desired, a value for the optional RUN_SLOW_FAC variable can be supplied through the PEST control file.

1.7 Parallel Run Queue Files

While PEST_HP is running, and after it has completed execution, you may see files named *pest###.dap* and *pest###.dao* in its working folder. These files may be very large. The PEST_HP run manager uses these files for parallel run management. *pest###.dap* contains parameter sets that are employed in a batch of model runs; a parameter set is transferred by the run manager to one of its agents whenever a model run is commissioned. *pest###.dao* contains model-generated observations corresponding to these parameter sets. These are sent by run agents back to the parallel run manager as model runs are completed.

The PEST_HP run manager does not delete these files on termination of PEST_HP execution in case their contents are required for a restart.

1.8 When Not to Use PEST_HP

As was discussed above, PEST_HP cannot undertake serial model runs; runs are always undertaken using a run agent. There is no lower limit on the number of run agents that PEST_HP can use; it can use as few as 1. However its inversion algorithm works best where there are many parallel run agents at its disposal. Ideally there should be at least ten (but hopefully many more), as its performance can be degraded when it works with fewer agents than this.

As presently programmed, the upper limit on the number of PEST_HP agents is set to 2048. If required for a specific application, this can be altered with ease; contact the programmer.

1.9 The "PEST Whisperer"

A program named PWHISP_HP is supplied with PEST_HP. PWHISP_HP reads PEST_HP input and output files, reflects upon what it sees in those files, and records all of its thoughts in a run record file of its own which can be inspected using a standard text editor. PWHISP_HP looks for any elements in PEST_HP setup that may compromise its efficiency, or impair its ability to perform highly parameterized inversion. It suggests to the user ways in which he/she can alter a PEST_HP input dataset, or modify PEST_HP deployment, in order to obtain better results. It tries to answer questions which a modeller may ask during or after a PEST_HP run, especially if he/she is wondering whether PEST_HP could have performed better than it did.

While much of the advice provided by PWHISP_HP is just as applicable to PEST as it is to PEST_HP, PWHISP_HP can only be used with PEST_HP. It cannot be used with PEST or BEOPEST or with any other member of the HP suite. Because of novel ways in which PEST_HP uses the Marquardt lambda and performs Broyden Jacobian updating, certain aspects of the run record file written by PEST_HP are different from that written by PEST. PWHISP_HP reads the PEST_HP run record file in detail; this is how it makes its assessment of PEST_HP performance. It has not been programmed to read a run record file written by any program other than PEST_HP.

PWHISP_HP is run using the command

```
pwhisp_hp case outfile
```

where *case.pst* is a PEST control file and *outfile* is the name for the text file which PWHISP_HP must write. This command can be issued even while PEST_HP is running; you do not have to wait until PEST_HP execution is complete in order to receive advice from the PEST whisperer. If they are available, PWHISP_HP reads some or all of the following PEST_HP input/output files, recording any suggestions that it can make based on the contents of these files on its own run record file.

- the PEST control file (*case.pst*);
- the PEST run record file (*case.rec*);
- the parameter value file (*case.par*, or *base_case.bpa* if PEST_HP is conducting SVD-assisted inversion);
- the composite sensitivity file (*case.sen*);
- the intermediate residuals file (*case.rei*); and
- the Jacobian matrix file (*case.jco*).

Note the following.

- PWHISP_HP does not offer advice on PEST_HP execution if PEST is run in “pareto” mode. This is because PEST_HP’s tasks when running in this mode are very different from when running in “estimation” or “regularisation” modes.
- It is not the intention of PWHISP_HP to repeat suggestions and warnings provided by PESTCHEK. PESTCHEK should be used to check a PEST_HP input dataset before PEST_HP is run. PWHISP_HP seeks verification from the user that this has indeed been done.

1.10 The PEST_HP Cost Estimator

Prior to running PEST_HP on the cloud, costs can be estimated using a utility program named PCOST_HP. PCOST_HP reads a PEST control file to obtain details of a PEST run; it obtains the other information that it needs for cost estimation from the command line. It is run using the command

```
pcost_hp case runtime agents rate [iteration_override]
```

where

case	is the filename base of a PEST control file;
runtime	(a real number) is the estimated model run time in minutes;

agents	(an integer) is the number of agents that PEST_HP will employ;
rate	(a real number) is the cost per minute per agent for cloud rental; and
iteration_override	(an optional integer) allows you to over-ride PCOST_HP's estimate of the number of iterations that PEST_HP will undertake.

Note that the cost per agent will be less than the cost per rented machine if a number of agents use the same machine. In fact the cost per agent will be the cost per machine divided by the number of agents that use each machine.

PCOST_HP lists the outcomes of its calculations to the screen. Figure 1.2 shows an example.

```
 Estimated cost without hp starter file = $29.00
 Estimated cost with hp starter file    = $27.20

 Some specifications:-
   PEST mode of operation                                 : regularization
   Number of adjustable parameters                        : 10
   Jacobian runs per itn before switch to higher order derivs : 10
   Jacobian runs per itn after  switch to higher order derivs : 20
   Parameter upgrade testing runs per iteration           : 5
   Number of upgrade testing cycles per iteration         : 1
   Number of iterations used for cost estimation          : 10
   Number of iterations specified in PEST control file    : 30

 Possible idle agents per iteration (in units of model runs):-
   Initial model run (if not using hp starter file)       : 4
   Initial model run (if using hp starter file)           : 0
   Fill Jacobian matrix before switch to higher order derivs  : 0
   Fill Jacobian matrix after  switch to higher order derivs  : 0
   Parameter upgrade testing                              : 0
```
Figure 1.2 An example of PCOST_HP screen output.

Cost estimates provided by PCOST_HP are only approximate. It has no idea how many iterations will be required for completion of the inversion process. So it guesses that the inversion process will reach completion after 8 iterations, unless the NOPTMAX variable in the PEST control file is set to less than this, or unless the setting of the SOFTSTOPHOURS or HARDSTOPHOURS variable (see later in this document) imposes an upper time limit on a PEST_HP run. PCOST_HP's calculations are also based on the assumption that the model run time will not vary from the initial estimate provided on its command line. In practice this may not be the case, especially for highly nonlinear models which employ adaptive time stepping for which the run time may be sensitive to the values of model parameters.

Costs may be considerably greater than estimated by PCOST_HP if PEST_HP is asked to run in "regularisation" mode, the number of parameters is high, and many of these parameters hit their bounds. As is explained in documentation of PEST, this situation requires many re-formulations of the inverse problem as pertinent parameters are sequentially frozen at their bounds. Implementation of the numerical steps required to accommodate bounds enforcement may keep the PEST_HP manager busy while agents are standing idle (sometimes for up to an hour if parameter numbers are greater than 10000, observation numbers are high, and many parameters encounter their bounds). PEST_HP cannot predict this. However it does issue a warning message if adjustable parameters number over 2500 and LSQR is not being employed as a solution mechanism for the inverse problem. (Note that, as is explained above, the time taken by PEST_HP to perform numerically intensive tasks can be considerably shortened if the *pest_hp_mkl.exe* executable program is used instead of the *pest_hp.exe* executable program. The time required to compute a regularisation weight factor can also be considerably reduced if the REG2MEASRAT variable is used to control regularisation; see section 7.8 of this document.)

A warning message is also issued if the setting of the HARDSTOPHOURS or SOFTSTOPHOURS variable affects calculation of the estimated PEST_HP running cost. Warning messages are placed under the cost estimate. See figure 1.3.

```
Estimated cost without hp starter file = $120.60
Estimated cost with hp starter file    = $119.60

Warning: estimated costs are limited by HARDSTOPHOURS setting in PST file.

Warning: the manager's run time (and hence the total cost) is unpredictable
as a large number of parameters is being estimated, but LSQR is not being
used as the solution mechanism.

Some specifications:-
  PEST mode of operation                                 : regularization
  Number of adjustable parameters                        : 1100
  Jacobian runs per itn before switch to higher order derivs : 1200
  Jacobian runs per itn after  switch to higher order derivs : 2400
  Parameter upgrade testing runs per iteration           : 100
  Number of upgrade testing cycles per iteration         : 1
  Number of iterations used for cost estimation          : 8
  Number of iterations specified in PEST control file    : 30

Possible idle agents per iteration (in units of model runs):-
  Initial model run (if not using hp starter file)       : 99
  Initial model run (if using hp starter file)           : 0
  Fill Jacobian matrix before switch to higher order derivs  : 0
  Fill Jacobian matrix after  switch to higher order derivs  : 0
  Parameter upgrade testing                              : 0
```

Figure 1.3 A cost estimate affected by the HARDSTOPHOURS setting, with an accompanying warning message pertaining to use of a large number of parameters.

As was stated above, PCOST_HP does not necessarily assume that the number of iterations required for completion of an inversion process is equal to the value of the NOPTMAX variable supplied in the "control data" section of the PEST control file. NOPTMAX is often set to a high value in order to allow other control variables to establish inversion completion. PCOST_HP assumes that after 8 iterations have elapsed, the objective function will have been reduced to a value that satisfies the user (in which case he/she will stop execution of PEST_HP him/herself), or to a value that PEST_HP cannot improve (in which case PEST_HP ceases execution itself). If neither of these occurs, then the NOPTMAX setting prevails, of course. The user can provide a value other than 8 for the purpose of cost estimation by entering this number as the value of the optional *iteration_override* command line variable. However PCOST_HP will reduce *iteration_override* to NOPTMAX if it is greater than NOPTMAX.

If PEST_HP is run in "pareto" mode, PCOST_HP does not assume that 8 iterations will be required for completion of the inversion process because, under these circumstances, PEST_HP traverses the Pareto front rather than undertaking a single inversion exercise. Instead, PCOST_HP assumes that PEST_HP will complete its traversal of the Pareto front, and that the number of iterations will therefore be equal to:

(NUM_WTFAC_INC-2)*NUM_ITER_GEN+NUM_ITER_START+NUM_ITER_FIN

(See the PEST manual for definitions of these Pareto control variables.) You can override this number through selection of a value for *iteration_override* which is less than this.

Note the following.

- If PCOST_HP encounters an error in the PEST control file which it reads, it ceases execution after writing an appropriate error message to the screen.
- An entry of “na” for any specification quoted by PCOST_HP means “not applicable”. The following conditions (and others) may result in a “na” condition being recorded in PCOST_HP’s screen output:
    - no parameter upgrade calculations are performed because NOPTMAX in the PEST control file is set to -2;
    - the switch to higher-order derivatives is not made because there is no parameter group for which the setting of FORCEN is either “switch” or “switch_5”.
- As is explained later in this document, the number of cycles of parameter upgrade testing per PEST_HP iteration can be reduced from two to one by disabling Broyden Jacobian improvement; this is achieved by setting the JACUPDATE variable to zero in the “control data” section of the PEST_HP control file.
- Use of a hp starter file is explained later in this document.

2. Marquardt Lambda Selection

2.1 The Selection Algorithm

In BEOPEST, implementation of the “partial parallelization” procedure whereby different values are selected for the Marquardt lambda, and parameter upgrades are calculated and tested using these different Marquardt lambdas, is dependent on the setting of the PARLAM parallelization control variable. According to this setting, the Marquardt lambda selection and parameter upgrade testing procedure can be serial, partially parallel, or fully parallel. Unless PARLAM is set to -9999, an initially parallel procedure can become a serial procedure if one or more parameters encounter their bounds.

The Marquardt lambda selection and parameter upgrade testing procedure undertaken by PEST_HP is always parallel; it never becomes serial, even if some parameters hit their bounds. On all iterations of the inversion process, PEST_HP selects a set of Marquardt lambdas, calculates parameter upgrades using these lambdas, and commissions a set of parallel model runs based on these upgraded parameter sets. If it has enough run agents at its disposal, it also calculates parameter upgrades corresponding to fractional lengths along the upgrade directions defined by different Marquardt lambdas. These fractions can be both greater and less than 1.0, with a fraction of 1.0 corresponding to the optimum length of the parameter upgrade vector according to an assumption of local model linearity. The line search fractions employed by PEST_HP are pre-set; however, on any particular iteration of the inversion process, PEST_HP can adjust them to reflect the distance in parameter space between current parameter values and the closest parameter bound in the direction of each upgrade vector.

Upgraded parameter sets calculated in the manner discussed above are bundled into a set of model runs that are undertaken in parallel. The multiplicity of upgraded parameter sets that require testing occupies computing cores that would otherwise be idle. PEST_HP ensures that the number of parameter upgrades that requires testing is such that the set of parallelized models runs which implement this testing can be undertaken simultaneously in roughly the same time (based on PEST_HP’s assessment of the relative computing speeds of agents which are at its disposal). Once these model runs have been completed, PEST_HP processes the results, and selects the best parameter set for potential use in the next iteration of the inversion process. (Note that “best” depends on whether PEST_HP is running in “estimation”, “regularisation” or “pareto” mode.)

Optionally, PEST_HP then repeats this process. However before doing so, it improves the Jacobian matrix using a Broyden update procedure informed by the results of the just-commissioned set of parallel model runs. The number of Broyden Jacobian updates which PEST_HP undertakes is set by the JACUPDATE variable supplied in the PEST control file; however it will never exceed the number of parameter upgrade directions determined by the number of Marquardt lambda values just employed (each Marquardt lambda defines a different update direction). Once this new set of parallelized model runs is complete, PEST_HP moves on to the next iteration.

During any iteration of the inversion process, PEST_HP calculates Marquardt lambda values itself. Its choice of the number of lambda values to employ depends on the number of available parallel run agents. However it will never employ less than three Marquardt lambda values, even if this is less than the number of parallel agents available to it. Furthermore, a user can constrain PEST_HP's choice of the number of Marquardt lambdas to employ in calculating upgrades through use of the PEST_HP-specific UPTESTMIN and UPTESTLIM control variables; see later in this manual. In calculating lambda values, PEST_HP ignores variables which control the Marquardt lambda selection procedure provided on the sixth line of the PEST control file. In particular, it ignores the RLAMDA1, RLAMFAC, PHIRATSUF, PHIREDLAM and NUMLAM variables. To make its indifference to the values supplied for these variables explicit, it does not even record them on its run record file. (It is nevertheless wise to choose sensible values for these variables so that PESTCHEK does not complain if asked to review the PEST control file.)

The optional JACUPDATE variable also appears on the sixth line of the PEST control file. If this is set to zero, or omitted from the PEST control file altogether, PEST_HP will not undertake Broyden enhancement of the Jacobian matrix based on the parallelized set of model runs which were carried out to test parameter upgrades. Instead, PEST_HP immediately commences the next iteration of the inversion process wherein it initiates finite-difference calculation of a new Jacobian matrix. Alternatively, if JACUPDATE is set to *N*, then PEST_HP undertakes *N* updates of the Jacobian matrix, based on directions in parameter space corresponding to the *N* most productive Marquardt lambdas. (Updating of the Jacobian matrix in this fashion does not require any model runs. It is an internal numerical procedure that is accomplished quickly; see PEST documentation for details.) If necessary, PEST_HP reduces *N* to the number of Marquardt lambdas that was actually used during the immediately preceding round of parallelized parameter upgrade tests. If you do not know what value to supply for JACUPDATE, set it to a high number such as 999; PEST_HP will then undertake as many Jacobian updates as it can. Alternatively, omit it from the PEST control file in order to forego Jacobian updating altogether. Limited experience to date suggests that on some occasions (particularly when running PEST_HP in "pareto" mode), the second round of parallelized parameter upgrade testing based on a Broyden-improved Jacobian matrix can be worth the trouble. On other occasions it is not, particularly if the number of available parallel run agents is large; in this case there is a greater computational advantage to be gained in re-computing the Jacobian matrix altogether using the just-upgraded parameter set than in trying to improve it in the hope of calculating a better upgrade.

PEST_HP will never undertake more than two rounds of parallelized parameter upgrade testing (and hence one round of Broyden Jacobian updating). Unless there are fewer than three agents at its disposal, each set of parallelized model runs required for parameter upgrade testing will be comprised of model runs which are fewer or equal in number to the number of agents which are at its disposal. The procedure is thus designed to maximize use of otherwise idle computing cores while progressing the inversion process as quickly as possible.

2.2 Upper Upgrade Test Limit

As stated above, when calculating parameter upgrades, PEST_HP selects Marquardt lambdas itself. It also calculates intervals along the parameter upgrade vectors that are computed using these Marquardt lambdas for which new parameters will be generated and then tested using parallelized model runs. The total number of upgraded parameter sets for which model runs can be commissioned can be as high as about 130, the exact number depending on the case. However, in practice, the number of parameter upgrades that are tested is restricted to the number of active agents. If more than 130 agents are available, then those agents which are not engaged in undertaking model runs for the purpose of testing parameter upgrades remain idle during the parameter upgrade testing procedure.

Its capacity to simultaneously test many different parameter upgrades is one of the strengths of the PEST_HP inversion algorithm. However, for some models, the number of runs devoted to upgrade testing may need to be restricted to a smaller number than PEST_HP would normally choose as it attempts to keep all agents busy. On some occasions, for some highly nonlinear models, parameter sets that are calculated using very high or very low Marquardt lambdas can induce convergence difficulties in the model's solver. Model instances which employ these parameter sets may then take a long time to run, this holding up the whole inversion process as other agents remain idle while waiting for these runs to finish. While this problem can be addressed to some extent using the RUN_ABANDON_FAC and WIN_MRUN_HOURS variables (see later in this manual), its chances of occurrence can be reduced by limiting the number of agents that are actually used in a single round of parallel upgrade testing. This limit applies to both the first round of parameter upgrade testing undertaken immediately after filling the Jacobian matrix, and to the second round of parameter upgrade testing that is commissioned after Broyden Jacobian updating has been completed. Upgrade agent limiting can be accomplished using the UPTESTLIM control variable.

The UPTESTLIM variable is optional. If present, it is placed in the "control data" section of the PEST control file – on the sixth line of this file. It can be placed anywhere after the values supplied for RLAMBDA1, RLAMFAC, PHIRATSUF, PHIREDLAM, NUMLAM, and (optionally) JACUPDATE, which also appear on this line. Suppose that you wish to limit PEST_HP to the use of no more than 50 agents when testing parameter upgrades. Then the string "UPTESTLIM=50" should be written on this line. See appendix 1 of this document for an example.

If the number of model runs devoted to parameter upgrade testing is limited in this fashion, then PEST_HP calculates Marquardt lambdas and line search factors in a way that maximizes upgrade testing efficiency while respecting this model run limit. However caution should be exercised in setting an upgrade testing limit in this way. If it is set too low, then some of the benefits of using PEST_HP will be lost.

2.3 Lower Upgrade Test Limit

Sometimes it is useful to set a lower limit on the number of parameter upgrade tests that PEST_HP undertakes, even if it does not have enough agents at its disposal to undertake the necessary model runs in parallel all at once. A user may insist, for example, that at least 20 parameter upgrade tests be undertaken even though PEST_HP may have access to 10 CPU's. He/she may do this in order to gain the benefits of the line search that PEST_HP undertakes along different parameter upgrade directions that are calculated using different Marquardt lambdas. This may prove a fruitful strategy in highly nonlinear inversion contexts.

The UPTESTMIN control variable can be used to set this minimum number of upgrade tests. Its use is almost identical to that of UPTESTLIM. Insert the string "UPTESTMIN=*N*" (where *N* must be replaced by the desired minimum number of upgrade tests) on the sixth line of the PEST control file anywhere after the values supplied for RLAMBDA1, RLAMFAC, PHIRATSUF, PHIREDLAM, NUMLAM, and (optionally) JACUPDATE.

Note that parallelization efficiency is maximized if UPTESTMIN is an integral multiplier of the number of agents that PEST_HP is using to supervise model runs.

3. HP Starter File

3.1 The Role of the Starter File

The serial version of standard PEST (including I64PEST, a 64 bit version of PEST) can now be run with a new command line option. This option is "/hpstart". Let us suppose that a PEST control file is named *case.pst*. Then to run PEST with this new option, use the command

```
pest case /hpstart
```

If initiated using this option, PEST runs the model only once using parameter values supplied in the PEST control file; it then ceases execution. As well as its usual set of output files, PEST writes a binary file named *case.hp*; this is referred to herein as a "hp starter file". This file is used by PEST_HP. If PEST_HP is to be run on another machine, or in the cloud, this file should be placed in the folder used by the PEST_HP manager, and in folders used by all run agents.

When execution of the PEST_HP manager is initiated, the following command can be used.

```
pest_hp case /hpstart /h :4004
```

When the PEST_HP manager is started in this way, it reads file *case.hp*, along with its normal set of PEST input files (i.e. the control, template and instruction files). The information contained in file *case.hp* eliminates the requirement for PEST_HP to undertake an initial model run in which it calculates values for model outputs using parameter values supplied in the PEST control file. (These outputs are used as reference values in finite-difference calculation of derivatives that are used to fill the first Jacobian matrix.) Instead, PEST_HP obtains these reference values from the hp starter file. These reference model outputs are also used to calculate the initial objective function, which is reported to the screen and to the run record file.

Because the PEST_HP manager obtains initial reference values from the hp starter file, and because the need for a model run based on initial parameter values is thereby eliminated, PEST_HP can commence finite-difference calculation of the initial Jacobian matrix immediately upon commencement of execution. All run agents can therefore be immediately put to work; there is no "gap time" in which they are waiting for completion of the initial model run (and thereby losing money if cores have been purchased on the cloud). This is a better option for handling of the initial model run than that provided by the "/p1" command line switch which is supported by BEOPEST, but not by PEST_HP. The "/p1" switch bundles the initial model run with the model runs required for filling of the first Jacobian matrix. Hence this first bundle of runs is comprised of *N*+1 model runs, where *N* is the number of adjustable parameters. Filling of the second Jacobian matrix requires only *N* model runs however, as reference observations used in finite-difference approximations to derivatives are calculated during the preceding Marquardt lambda selection and parameter upgrade testing process. Where inversion takes place in the cloud, this disparity between the number of model runs required to fill successive Jacobian matrices can make selection of an optimal number of processors difficult. Ideally *N* should be an integer multiple of the number of available processors; no processors are therefore idle as model runs are undertaken for filling of the Jacobian matrix.

(Note that the above logic assumes that one model run is required per adjustable parameter. If derivatives with respect to all parameters are calculated using higher order finite-differences, the same logic applies, as the number of model runs required for filling the Jacobian matrix remains an integer multiple of *N*. The argument breaks down, however, where forward differences are used for calculating derivatives with respect to some parameters and higher order differences are used for calculating derivatives with respect to other parameters. It is also invalidated where PEST's observation re-referencing functionality is invoked. This does not erode the value of the hp starter file however.)

The PEST_HP run agent (i.e. AGENT_HP) also reads the hp starter file. However it does not need to be started using the "/hpstart" command line switch; in fact if this switch is provided to AGENT_HP, it is ignored (as are any other command line options). Instead, AGENT_HP is informed of the fact that it should read a hp starter file by the PEST_HP run manager (if its execution is initiated using the "/hpstart" command line option). The contents of the hp starter file include information which can facilitate the reading of lengthy model output files by AGENT_HP. In particular, AGENT_HP is informed of the order in which observations are encountered as model output files are read. Where observations number in the hundreds of thousands, and/or where ordering of model outputs corresponding to observations does not coincide with the ordering of observations in the PEST control file, this can reduce the time to read model output files considerably on the first occasion on which they are read by AGENT_HP. (Note that if a hp starter file is not employed, then AGENT_HP orders observations internally as it reads model output files for the first time; its second reading of these files is therefore fast, regardless of the presence, or otherwise, of a hp starter file.)

If either PEST_HP or AGENT_HP is asked to read a hp starter file, but the file is not actually present in the folder from which it is run, the absence of this file will be tolerated. PEST_HP will commission the first model run using a single agent, just as if it had not been requested to read a hp starter file at all. If AGENT_HP does not find an expected hp starter file in its folder, it will read model output files in the usual way, without the benefit of ordering information contained in the hp starter file. Unless observation numbers are very large, this is not a serious disadvantage. Where observation numbers are very large, the benefits of ordering are available on subsequent occasions that AGENT_HP reads model output files.

A hp starter file must be written by PEST or I64PEST. It cannot be produced by BEOPEST. A serial version of PEST must be used to produce a hp starter file because it stores both reference observations and the ordering of observations in model output files. Where an initial model run is undertaken by BEOPEST, the former information is available to the manager whereas the latter information is obtained by an agent. It is therefore difficult to record both sets of information in the same file. Use of a serial version of PEST to write the hp starter file is not a constraint however, as the concept of "parallel" makes little sense when only one model run is undertaken.

If execution of PEST_HP is re-commenced using the "/s" switch, then the presence of a "/hpstart" switch is ignored – except for the fact that the PEST_HP manager instructs its agents to read this file in order to accrue the efficiency gains discussed above where observation numbers are large.

3.2 The "/i" Command Line Option

Execution of PEST_HP can be initiated using the "/i" command line switch. (This switch is also available for PEST and BEOPEST.) If started in this way, PEST_HP requests the name of a JCO file which it must read to obtain the initial Jacobian matrix; it does not therefore need to fill this Jacobian matrix using finite-difference-calculated derivatives. See PEST documentation for further details.

If execution of PEST_HP is initiated using both the "/hpstart" and "/i" command line options, then the need to undertake both the initial model run, and any model runs required for filling of the initial Jacobian matrix, are eliminated. PEST_HP can therefore commence calculation of parameter upgrades using different Marquardt lambdas immediately. However it has been programmed to pause for about a minute before commencing this process; this provides time for all run agents to connect to the manager. As has been discussed above, the Marquardt lambda and line search strategy adopted by PEST_HP for calculation of parameter upgrades depends on the number of agents to which it has access.

Note that, as stated above, AGENT_HP ignores all command line options (except the "/h" option through which it learns the contact details of its manager).

3.3 SVD-Assist and the "/hpstart" Option

There is nothing to be gained by PEST_HP reading a hp starter file if SVD-assisted inversion is being undertaken. The Jacobian matrix file produced by PEST (or by BEOPEST or PEST_HP) as a precursor to initiation of SVD-assisted inversion actually includes reference values for observations. Neither the initial model run, nor model runs required for filling of the first Jacobian matrix, need therefore be undertaken by either PEST_HP or by PEST/BEOPEST at the commencement of an SVD-assisted inversion process. Use of the "/hpstart" command line switch therefore has no effect on the operation of PEST_HP, except for the fact that it informs its run agents to expect a hp starter file in their working folders. As is explained above, the agents can then use this file to order observations prior to reading model output files so that the reading of these files can be expedited.

4. Run Results File

4.1 File Contents

As the name implies, a "run results file" records the outcomes of a sequence of model runs. The first line of the file provides the number of parameters and number of model outputs (i.e. observations) which are featured in the file. These coincide with the number of parameters and observations featured in the PEST control file on which a PEST_HP run is based. The run results file then records the names of all of these. Then, for a sequence of model runs, it provides parameter values used for those runs together with model outcomes (i.e. the model-generated counterparts to observations featured in the PEST control file) calculated on the basis of these parameters. (Note that the outcomes of prior information equations are not reported.) The run results file is an ASCII file. Its format is obvious from an inspection of it.

A run results file can serve many purposes. These include uncertainty analysis (where parameter values are randomly selected, or semi-randomly selected using methodologies such as Latin hypercube), global sensitivity analysis (where parameter values follow a Sobol sequence or express trajectories defined by sequential variation of individual parameters), and construction of a proxy model that can partially replace a large, complex model in calibration of the latter.

4.2 The RRFSAVE Option

There are two means by which PEST_HP can be instructed to write a run results file. The first is through use of the RRFSAVE variable in the "control data" section of the PEST control file. This variable appears on the tenth line of this file; it must follow the values of the ICOV, ICOR, IEIG and optional IRES variables (these variables are ignored by PEST_HP as it does not write the corresponding files). It can be supplied anywhere following these variables, intermixed with other variables that can also optionally appear on this line (namely VERBOSEREC, JCOSAVEITN, REISAVEITN, PARSAVEITN and PARSAVERUN); see appendix 1 for an example. If the value of the RRFSAVE variable is supplied as "rrfsave", then PEST_HP will record a run results file. Alternatively, if RRFSAVE is supplied as "norrfsave", or omitted altogether, then PEST_HP will not record a run results file.

If recording of a run results file is instigated in this manner, then PEST_HP does not record run results for every single run which it commissions. In particular, results are not saved for model runs which are undertaken for the purpose of finite-difference derivatives calculation because

1. Too many of these runs are undertaken; and
2. Parameter values are very similar for these runs.

Instead it records outcomes of the following runs.

1. The initial model run;
2. All model runs undertaken for testing of parameter upgrades.

If the name of the PEST control file is *case.pst*, then the run results file saved by PEST_HP is named *case.rrf*.

In complying with the protocol of a run results file, PEST_HP records the reason for which each recorded model run was carried out. Where applicable, this includes the value of the Marquardt lambda and line search factor, and whether or not Broyden Jacobian improvement was being implemented. This is followed by the values of all parameters employed for that model run, and the values of model-calculated observations read from model output files recorded on that model run.

In the event of a failed model run, all model output values for that run are recorded as -1.11E35. In contrast, model output values of -1.22E35 indicate an abandoned model run.

4.3 The “/f” Command Line Option

If started with the “/f” command line switch, PEST_HP does not undertake parameter estimation at all. Instead it undertakes a series of model runs specifically for the purpose of calculating model outputs using different sets of parameters. These parameter sets must be supplied in a sequence of parameter value files recorded prior to running PEST_HP. They can be constructed using PEST-suite utility programs such as RANDPAR, RANDPAR1 and LHS2PEST; a Sobol sequence parameter set generator is available on request. The protocol for parameter value files is such that a single set of parameters is stored in each file. (See PEST documentation for specifications of this type of file.) PEST_HP expects that these files are named as a sequence with a common filename base. Suppose that the filename base is *sample*. Then the parameter value files must be named *sample1.par*, *sample2.par*, *sample3.par*, etc. The sequence does not need to begin at 1; however there must be no gaps in the sequence. The names of the parameters which appear in each of these files must be the same. They must correspond to parameters that appear in the PEST control file on which basis PEST_HP is run.

Suppose that PEST_HP is started using the following command.

```
      pest_hp case /f /h :4004
```

Then, upon commencement of execution, PEST_HP reads the PEST control file *case.pst*. It then issues a series of prompts. These prompts, and possible responses to these prompts, are presented below.

```
PEST_HP will run the model repeatedly, using parameter values recorded
 in a sequence of parameter value files. It will record all model-calculated
 observations in a run results file.

 Enter filename base of parameter value files: parval
 Enter first index to use: 101
 Enter last index to use: 200

 Enter parallel run packet size: 50

 Enter name for run results file: record.rrf
```

Responses to the above prompts instruct PEST_HP to read files *parval101.par* to *parval200.par* to obtain 100 sets of parameter values. (If the names of parameters read from these files do not correspond to parameter names recorded in the PEST control file *case.pst*, PEST_HP will cease execution with an appropriate error message.) The PEST_HP run manager will then tell its agents to carry out model runs based on these parameters. In the above example, parallel model runs will be carried out in packets of 50. (It is the user’s responsibility to choose the run packet size in a way that minimizes the chances of agents being idle; the number of agents available for carrying out model runs in this example should thus be 5, 10, 25 or 50.) After each packet of model runs has been completed, PEST_HP calculates objective function components and writes these to the screen and to its run record file; it also records parameter values and model outputs for all runs comprising the packet in the run results file. It then initiates the next packet of parallel model runs.

There is no reason why the run packet size cannot be equal to the total number of parameter sets for which model outputs must be calculated. This is a matter of convenience. Sometimes a useful choice for run packet size is a number which is a small multiple of the number of available run agents. This allows you to view objective functions before the complete set of model runs is finished, thus allowing verification that the model's behaviour is in accordance with expectations. On the other hand, if model run times are sensitive to parameter values, some agents may intermittently become idle while waiting for a packet of model runs to finish because of slow model run times on other agents. Maximum efficiency is thereby gained if the run packet size is equal to the number of runs which must be undertaken.

In the event of model run failure, PEST_HP does not cease execution with an error message if started with the "/f" switch. Nor does it try to repeat a failed model run using another agent. Instead it assumes that the model did not like the parameter set with which it was provided. The objective function is recorded as -1.11E35. All model outputs associated with the offending parameter set are recorded in the run results file as -1.11E35. Model run failure is thus easily recognized.

If an appropriate value is supplied for the RUN_ABANDON_FAC variable in the "control data" section of the PEST_HP control file (see section 7.2.3), PEST_HP will abandon model runs that are requested in any run packet if these runs are unduly late. The role of the RUN_ABANDON_FAC variable in defining "late" when PEST_HP is run with the "/f" command line option is the same as its role in defining "late" where model runs are undertaken for parameter upgrade testing. Model outputs of abandoned runs are all assigned a value of -1.22E35 prior to recording these values in the run results file. The abandoned status of the model run is recorded in the header to the subsection of the run results file in which the results of the abandoned model run are recorded. Meanwhile values of -1.22E35 are recorded for the total objective function, and components of the objective function pertaining to the abandoned model run, in the PEST run record file.

A non-zero WIN_MRUN_HOURS setting (see section 7.2.4) is also respected if PEST is run with the "/f" switch. Agent-terminated model runs are treated as failed model runs. Objective function and model output values recorded in the run record and run results files reflect this status.

As is documented in the PEST manual, a parameter value file supplies values for the PRECIS and DPOINT variables used by PEST for writing numbers to model input files; it also provides the SCALE and OFFSET for each parameter. PEST_HP ignores all of these in reading the sequence of parameter value files to which it is directed when run with the "/f" command line option. Instead, values for all of these variables are obtained from the PEST control file. This frees the parameter value generator which writes the parameter value files from needing to know the values of these control variables.

When the PEST_HP manager is run with the "/f" switch, each agent writes a file named *pest####.pfl* in its working directory on each occasion that it runs the model. This file contains a single entry, namely the name of the parameter value file from which the manager obtained parameter values that pertain to the current model run. If the parameter value filename contains a blank, its name is enclosed in quotes; otherwise it is not. This file is overwritten on each occasion that the agent runs the model.

4.4 Reading a Run Results File

The RRFCALCPSI utility supplied with the standard PEST suite can read a run results file calculated by PEST_HP. It calculates objective function components associated with each observation group. However it adds value to the contents of the run results file by allowing a different PEST control file to be used in calculation of objective function values from that which was used by PEST_HP in the production of this file.

See Part II of the PEST manual for details.

5. Model File Distribution

5.1 General

Suppose that a modeller is calibrating (for example) a steady state groundwater model, or a natural state geothermal reservoir model. The execution speeds of models such as these are much greater if initial pressures provided to the model are close to final pressures. The latter depend, of course, on parameter values. It is PEST_HP's job to adjust these such that model-generated pressures and/or flows (calculated from pressures) match observed pressures and/or flows at measurement sites.

During each iteration of the inversion process through which a model such as this is calibrated, PEST_HP attempts to improve the fit between model-calculated pressures/flows and their measured counterparts. After finite-difference derivatives have been calculated, different parameter upgrades, calculated using different Marquardt lambdas, and different line search fractions along different upgrade directions, are tested on the different computing nodes on which PEST_HP agents are running. The model that is run on one of these nodes employs the best parameters calculated for that iteration. PEST_HP does not know in advance which parameter set will be the best; this becomes apparent after the model which employs this parameter set, and models which employ other parameter sets, have been run by their respective agents, and objective functions associated with all of these parameter sets have been calculated.

The best parameter set achieved during any iteration of the inversion process is used as the starting point for the next iteration. Finite difference derivatives are calculated by varying parameters individually and incrementally from base values comprising this parameter set. Ideally, the initial pressures used by the model for all of these parameter-increment runs should be those calculated on the basis of the best parameter set from the previous iteration. The model runs devoted to finite-difference derivatives calculation should then be fast because solution pressures calculated for incremented parameters should be only incrementally different from initial pressures calculated using base parameters.

One way of achieving this desirable outcome is to employ PEST_HP's observation re-referencing functionality. Thus at the beginning of each new iteration, a special model run is undertaken in which best parameters achieved during the previous iteration are employed. Pressures calculated using this parameter set are then saved as initial pressures for an immediately ensuing model run. This next model run should take very little time; its purpose is to create reference model outputs used in finite-difference derivatives calculation. Meanwhile these same initial pressures should be distributed to every agent at PEST_HP's disposal so that model runs that are undertaken by these agents are also fast. The speed of the finite-difference derivatives calculation process is thus greatly increased.

Note how, in the above procedure, the simulator is run twice when PEST-HP undertakes the re-referencing model run. During both of these runs it employs best parameters from the previous iteration. The first simulation calculates steady-state/natural state pressures based on these parameter values; the second simulation uses these pressures for its initial conditions. In doing so it reproduces the simulation context in which other model runs will be undertaken with parameters incrementally varied from their current values. As stated above, pressures and flows at observation locations calculated during this second simulation become reference pressures used in finite difference derivatives calculation.

While this approach has been used successfully on many occasions, some problems may arise in its implementation – particularly in the cloud environment. One problem is that of distributing the file which contains initial pressures to agent nodes. The PEST_HP run manager node may not have permission to write to agent nodes. It may not even know where they are! Another problem is that this procedure requires that the model be run using best parameters achieved up to this point in the inversion process at the start of each new iteration (just before commencement of finite-difference derivatives calculation). However this same run has already been done during the previous iteration. Furthermore, all agents other than that which is undertaking the re-referencing run must stand idle while this repeated run proceeds. This does not constitute efficient usage of cloud resources.

These problems can be overcome by using PEST_HP's file distribution functionality. At the end of each iteration of the inversion process PEST_HP can optionally fetch one or a number of files from the node on which the best model run from the previous iteration was carried out. These files are first transferred to the PEST_HP manager's node using TCP/IP – the same protocol as that employed for all other PEST_HP manager-to-agent communication. Then, optionally, the manager can issue a user-specified system command to process these files; this processing is undertaken locally on its own node. Then calculation of finite-difference derivatives commences. However before PEST_HP commissions a model run on any particular agent, it transfers one or a number of files to that agent. These files can be the same as those previously transferred to the manager from the node on which the best model run was carried out. Alternatively, new files calculated by the manager node on the basis of these transferred files can be transferred to each agent node. As before, transfer between manager and agent (i.e. PEST_HP and AGENT_HP) is completed using TCP/IP. Thus write permission on agent nodes is not required. The model is then run by the respective incidence of AGENT_HP with a parameter incrementally varied in the usual manner.

This file distribution protocol eliminates the need for the manager to know where each agent is situated. It also eliminates the need for the manager to have read/write permission on each agent's drive. At the same time, this protocol reduces the computational burden of the re-referencing run. An expensive simulation with old starting pressures is no longer required. Instead, the re-referencing simulation can use pressures calculated during the parameter upgrade procedure of the previous iteration as its initial pressures. These same initial pressures are distributed to all agents for finite-difference derivatives calculation.

5.2 PEST Control File

To implement strategic file distribution as outlined above, a new section must be introduced to the PEST control file. This "distribution files" section must be the final section of the file. Figure 5.1 shows specifications of the "distribution files" section of the PEST control file while figure 5.2 shows an example.

```
* distribution files
DISTRIB_TYPE SOURCEFILE_AGENT TARGETFILE_MANAGER [SOURCEFILE_MANAGER TARGETFILE_AGENT]
The above line is repeated up to four times.
command = DISTRIB_MODEL_COMMAND
The above line is optional and can be inserted anywhere in the "distribution files" section.
```

Figure 5.1 Specifications for the "distribution files" section of a PEST control file.

```
* distribution files
2 model.out temp.dat init.dat init.dat
1 model.out init.dat
command = process_outputs.exe
```

Figure 5.2 Example of the "distribution files" section of a PEST control file.

As for any other section of a PEST control file, the "distribution files" section must begin with a header line which provides the name of the section. This line must begin with an asterisk character followed by a space.

Up to six lines of data can follow the "* distribution files" header. The section is considered to finish when a blank line, or the end of the PEST control file, is encountered. Only five lines can follow the section header if a model command is not supplied.

The optional command associated with the "distribution files" section (i.e. DISTRIB_MODEL_COMMAND) can be provided on any of the lines which follow the "* distribution files" section header. However only one such command can be supplied. The line which provides this command must commence with the string "command =". The actual command which the PEST_HP manager will deliver to the operating system must follow this string.

Note the following features pertaining to the protocol of the line which provides the optional system command associated with file distribution.

- The word "command" can be supplied in upper or lower case, or in a combination of the two.
- The "=" character which follows the "command" string can optionally be separated from this string by a space.
- The command which follows the "=" character can optionally be separated from it by a space.
- The text comprising the system command can optionally be surrounded by double quotes.

As for other commands cited in the PEST control file, the text comprising DSITRIB_MODEL_COMMAND of figure 5.1 will be provided to the operating system exactly as written by the user. This command can be the name of a batch file, or the name of an executable program; optionally it can also include command line arguments.

Lines other than that which provides the DISTRIB_MODEL_COMMAND which appear in the "distribution files" section of the PEST control file (there can be a maximum of five of these) must all begin with an integer. This integer must be 1 or 2. This is the value of the DISTRIB_TYPE variable associated with each distribution file.

If DISTRIB_TYPE is set to 1, then the nominated file on the node on which the best model run was achieved during the previous round of parameter upgrade testing is copied to the manager's node; however this file is not distributed to nodes used by other agents. The modeller must therefore organize file distribution to these nodes him/herself. Alternatively, in some parallelization contexts, all model incidences may read this file from the manager's working folder themselves, this obviating the need for distribution of the file to agent nodes. In contrast, if DISTRIB_TYPE is set to 2, the PEST_HP manager first copies the nominated file to its own node from the best-upgrade node, and then copies it to each agent node, possibly after processing the file using the DISTRIB_MODEL_COMMAND system command.

If DISTRIB_TYPE is set to 1, then two filenames must follow the value of this variable. These are denoted as SOURCEFILE_AGENT and TARGETFILE_MANAGER in figure 5.1. The first is the name of the file that must be copied from the best-upgrade agent’s node. The second is the name that will be given to this file on the manager’s node. If either of these filenames contain spaces, the filenames should be enclosed in quotes. These names can include (relative) pathnames if desired. In many cases the names given to SOURCEFILE_AGENT and TARGETFILE_MANAGER will be the same, and contain no pathnames; hence the file is simply copied from the working directory (i.e. folder) of the best-upgrade agent to the working directory of the PEST_HP manager.

If DISTRIB_TYPE is set to 2, then four filenames must follow the value of this variable. The first two are identical to those required when DISTRIB_TYPE is set to 1. However the other two are required for copying the file from the manager’s node to agent nodes. SOURCEFILE_MANAGER is the name of the file that must be copied from the manager node to all agent nodes. TARGETFILE_AGENT is the name given to this file on agent nodes once it has been copied. In many circumstances SOURCEFILE_MANAGER will be the same as TARGETFILE_MANAGER. However if a DISTRIB_MODEL_COMMAND is run by PEST_HP, then this command may process the TARGETFILE_MANAGER file to produce a new SOURCEFILE_MANAGER file that is appropriate for distribution to agents. While PEST_HP allows this complication, and allows a file copied from the best-upgrade agent’s node to be named differently when it appears on the manager’s node, there will be many occasions where such complexity is not warranted; there will also be many occasions where no local command is employed to process the copied file. In this case all of the filenames associated with a DISTRIB_TYPE of 2 will be the same.

5.3 Implementation Details

PEST_HP initiates file distribution at the end of each iteration of the inversion process, just before it commences the next iteration (and possibly at the end of the inversion process – see below). First, regardless of the DISTRIB_TYPE setting associated with each distribution file, PEST_HP copies the file from the best-upgrade agent’s node to the manager’s node. This is done for all distribution files cited in the “distribution files” section of the PEST control file. If this process fails for any of the distribution files, PEST_HP ceases execution with an appropriate error message. After it has transferred all distribution files to the manager’s node (recall that there can be up to five of these), PEST_HP issues the optional system command associated with its file distribution functionality (i.e. DISTRIB_MODEL_COMMAND of figure 5.1). It then proceeds to the next iteration.

For distribution files with a DISTRIB_TYPE of 2, PEST_HP does not transfer the pertinent distribution file to an agent’s node until just before the next occasion on which it employs that agent to carry out a model run. If there is a failure in this transfer, PEST_HP ceases execution with an appropriate error message.

Activation of file distribution functionality requires some small modifications to normal PEST_HP functionality in order to guarantee the integrity of the inversion process. These modifications are as follows.

- If, during any iteration of the inversion process, the objective function fails to fall, and if this failure triggers PEST_HP's initiation of higher order finite-difference derivatives calculation, PEST_HP will normally commence the new iteration using the best parameters that have been achieved so far during the inversion process, even though they were not achieved during the previous iteration. This strategy cannot be adopted if file distribution functionality is in place because model-generated files associated with the best parameter set will have been lost from the node which employed this parameter set for its model run. Instead, the next iteration employs the best parameter set from the previous iteration despite the fact that this is not the best overall parameter set achieved during the inversion process to date.
- There may be occasions when the PEST_HP run manager loses access to the node of the best-upgrade agent before it can copy files from that agent's node to its own node. (This can happen if the agent disappears from the network, or if PEST_HP has access to less than three nodes.) If this happens, PEST_HP records this state of affairs to the screen and to the run record file; it then continues to the next iteration without file distribution.
- All distribution file transfers are recorded on the run management record file.

Note also the following.

- If more than one file requires distribution, then PEST_HP will allow duplicate names for the SOURCEFILE_AGENT and SOURCEFILE_MANAGER files associated with these different distribution files (though it is unclear why a user would want to do this). However it will not allow duplicate names for the TARGETFILE_MANAGER or TARGETFILE_AGENT filenames associated with different distribution files.
- Even if no source and target files are provided in the "distribution files" section of the PEST control file, a DISTRIB_MODEL_COMMAND can still be supplied. If so, the PEST_HP manager will send this command to the operating system at the close of every iteration of the inversion process.
- PEST_HP cannot be restarted using the "/s" switch if file distribution functionality is activated, as it does not know the status of distribution files on either manager or agent nodes on recommencement of execution. Use the PARREP utility to create a new PEST control file based on optimised parameters from the previous run to restart it under these circumstances.
- PEST_HP cannot be started with the "/f" switch if file distribution functionality is activated as use of this functionality would support no gains in efficiency when PEST_HP is used in this way.
- If TCP/IP communications fail when an agent is receiving a distribution file from its manager, or if AGENT_HP cannot write the distribution file to its local drive, it ceases execution with an appropriate error message. The agent is therefore lost to the inversion process.

On termination of execution, PEST_HP copies distribution files from the folder of the best-upgrade agent to that of the manager

- if it can, and
- if parameters underwent improvement on the final iteration of the inversion process.

Success of this operation depends on whether the files that were written by the best-upgrade agent on the last parameter upgrade cycle are still present in that agent's folder. If PEST_HP is able to copy these files back to the manager, if notifies the user through its screen output, its run record file and its run management record file. If not, no message is recorded. In neither case is the DISTRIB_MODEL_COMMAND issued to the operating system.

5.4 File Distribution and Broyden Jacobian Updating

Notwithstanding the extra book-keeping that it requires, PEST_HP does not prevent Broyden updating of the Jacobian matrix if file distribution is activated. This is because conjunctive use of these two items of PEST_HP functionality may be warranted when calibrating large, highly parameterized, nonlinear models such as geothermal reservoir models.

Caution must be exercised in implementing both of these options together, however. In particular, it is advisable that the directory used by the PEST_HP manager should not also be employed by a PEST_HP agent.

During each iteration of the inversion process, pertinent files that are nominated in the "distribution files" section of the PEST control file are copied from the folder used by the agent that supervised the best model run, to the folder used by the PEST_HP run manager, at the end of the first parameter upgrade testing cycle, regardless of whether a second round of upgrade testing takes place following Broyden Jacobian updating. If a second round of upgrade testing does, indeed, take place, files earmarked for distribution are copied to the manager's folder only if the best parameter set obtained during that second testing cycle is an improvement over the best obtained during the previous cycle of parameter upgrade testing (i.e. the cycle of parameter upgrade testing undertaken without a Broyden-updated Jacobian matrix). File distribution from the manager's folder to agent folders (for files with a DISTRIB_TYPE of 2) then takes place as agent nodes are commissioned to undertake model runs at the beginning of the next iteration of the inversion process.

5.5 File Distribution and Randomized Jacobian

File distribution to agent node folders does not occur if PEST_HP is using a randomized Jacobian matrix (see section 9 of this document), and the covariance matrix from the previous iteration on which this Jacobian matrix is based is being supplemented with model runs from the present iteration in accordance with the setting of the NRANDREPEAT control variable. Under these circumstances, parameter reference values are unchanged for the next iteration. Hence files copied back to the manager's folder from the best-run agent's folder during the previous iteration cannot be used to provide initial conditions for model runs undertaken during the subsequent iteration of the inversion process as the upgraded parameters from the previous iteration have been rejected (because they did not result in objective function improvement). Instead, reference parameter values employed at the beginning of the previous iteration are retained for the new iteration.

Similar considerations apply to the optional DISTRIB_MODEL_COMMAND system command. Where PEST_HP is not using a randomized Jacobian matrix, this command is issued at the beginning of every iteration of the inversion process because upgraded parameters from the previous iteration are employed as reference values for the ensuring iteration, even if they did not lower the objective function. Where a randomized Jacobian matrix is employed, however, the DSITRIB_MODEL_COMMAND system command is not issued where a reference parameter set persists to the next iteration in accordance with the dictates of NRANDREPEAT functionality.

6. PEST_HP-Specific Output Files

6.1 General

PEST_HP records the same suite of output files that other versions of PEST record (with the exception of file *case.mtt* which provides a linear approximation to the posterior parameter covariance matrix in over-determined parameter estimation contexts). However it records some extra output files (in addition to the optional run results file described in the previous section); these are now described.

6.2 Objective Function Record File

Unless it is run using the "/f" command line switch, PEST_HP writes an "objective function record file" named *case.ofr* (on the assumption that the governing PEST control file is *case.pst*). This file contains a table which links the value of the objective function, and components thereof, to the iteration number of the inversion process. It is updated at the beginning of each iteration. This file is easily imported into a spreadsheet or graphing package for visual tracking of PEST_HP progress.

Where PEST_HP is run in "regularisation" mode, the objective function record file lists values for the measurement and regularisation objective functions, as well as for the objective function components associated with all non-regularisation observation groups. It does not provide objective function values for regularisation groups as these are non-comparable between iterations; they are subject to multiplication by the regularisation weight factor, and to inter-group weight factor adjustment as designated by the value of the IREGADJ regularisation control variable.

Where PEST_HP is run in "estimation" or "pareto" modes, the objective function record file lists values for the total objective function, and for the objective function components associated with all observation groups.

6.3 Parallel Run Efficiency File

Suppose that PEST_HP is run on the basis of the PEST control file *case.pst*. In common with BEOPEST, PEST_HP records parallel run management details in a file named *case.rmr* (the parallel run management record file). In contrast to BEOPEST however, PEST_HP records an additional "parallel run efficiency file" named *case.rme*. This file contains a table which can be easily imported into a spreadsheet or graphing package for plotting. For each parallel agent, the parallel run efficiency file records:

- the agent's working folder;
- the current status of the agent (active, idle or lost);
- the number of model runs carried out by the agent;
- the number of run failures encountered by the agent;
- the number of times that a run undertaken by the agent was abandoned or assigned to another agent;
- the number of successful model runs carried out by the agent;
- the maximum, minimum and average model run times (in seconds) encountered by the agent;
- the cumulative run time (in seconds) over all model runs undertaken by the agent;

- the cumulative time (in seconds) over which the agent has existed;
- the parallel run efficiency of the agent (calculated by dividing cumulative run time by cumulative existence time).

These statistics need some refinement for cases where different commands can be used to run the model in different circumstances, for example if multiple model commands are employed for the purpose of finite difference derivatives calculation, and/or if observation re-referencing is undertaken. This refinement awaits further PEST_HP development.

The parallel run efficiency file is updated every minute, or at the completion of every parallel package of model runs, whichever happens sooner.

PEST_HP issues a warning message to the screen at the beginning of each iteration of the inversion process if a particular agent has encountered run failures but no run successes. This may indicate that a file is missing or corrupted in that agent's folder.

6.4 Parameter Error File

See section 7.1.2.

7. New and Altered Control Variables

7.1 Model Run Failure

7.1.1 LAMFORGIVE and DERFORGIVE

If the LAMFORGIVE variable on the sixth line of the PEST control file is set to “nolamforgive”, or if this variable is omitted from the PEST control file, then BEOPEST will cease execution with an appropriate error message if an agent reports model run failure for a particular set of updated parameters. But first the BEOPEST run manager asks that the run be repeated by another agent; if failure is repeated, the manager ceases execution with the pertinent error message. Alternatively, if LAMFORGIVE is set to “lamforgive”, then BEOPEST will not repeat the offending model run. Instead it will declare that the objective function corresponding to the offending set of parameters is very high; this provides a signal to the inversion process that this part of parameter space is unproductive.

PEST_HP ignores the LAMFORGIVE setting (or absence of LAMFORGIVE setting) provided by a user in the PEST control file. It sets LAMFORGIVE internally to “lamforgive”.

PEST_HP does not ignore the setting of the DERFORGIVE variable provided in the PEST control file, however; instead, it treats this variable in the same way that BEOPEST treats it. If DERFORGIVE is set to “noderforgive” (or omitted from the PEST control file), then model run failure during filling of the Jacobian matrix causes PEST_HP to repeat the run and then, if failure occurs again, terminate execution with an appropriate error message. Alternatively, if DERFORGIVE is set to “derforgive”, PEST_HP does not repeat the failed model run using another agent. Instead it declares the sensitivities of all model outputs to the incremented/decremented parameter to be zero. It does not cease execution; it continues to fill the Jacobian matrix by processing other model run outcomes.

7.1.2 Record of Offending Parameter Sets

PEST_HP records all parameter sets which precipitate model run failure in a “parameter error file” named *case.per* (assuming that the PEST control file on which the current PEST_HP run is based is named *case.pst*). Offending parameter sets are listed using a format which is not unlike that of a parameter value file. Hence, with a little cutting and pasting, a parameter value file is readily built using any of the offending parameter sets recorded in a parameter error file. The PARREP utility can then be employed to populate a new PEST control file using these parameters. NOPTMAX can be set to zero in the new PEST control file; PEST can then be run to reproduce the problem.

When PEST_HP is employed to undertake SVD-assisted inversion then, as is documented in the PEST manual, the inversion process is based on so-called super parameters. In the event of model run failure, PEST_HP records the values of not only super parameters, but also of base parameters, in its parameter error file. The latter can be PARREPed into the base parameter PEST control file to reproduce the model run problem in the manner described above. It may not be possible to reproduce the problem by PARREPing super parameters into the super parameter PEST control file as definition of super parameters can change during the course of the inversion process.

7.2 Handling of Overdue Model Runs

7.2.1 General

Sometimes model runs can take longer than expected to reach completion. Where parallel model runs are distributed across a local network, this may happen because the computer on which a local agent is running has been given other work. Alternatively, it may be perceived that the model is taking a long time to run because network communications between the manager and the pertinent agent have broken down. A third alternative is that the parameter set on which the model run is based may be creating convergence difficulties for the model solver. The first two of these occurrences are unlikely in a cloud environment. The third can sometimes happen where model runs are being undertaken for the purpose of testing parameter upgrades. Under these circumstances the parameter sets on which the different model runs are based can be very different.

PEST_HP adopts two different strategies for the handling of late model runs; the strategy that is adopted on any particular occasion depends on the reason for which model runs are being undertaken. When derivatives are being calculated, PEST_HP re-assigns overdue model runs to alternative agents which have already completed the model runs assigned to them. (Note that model run re-assignment can be prevented, if desired, in ways described below.) On the other hand, where a packet of model runs is initiated for testing a variety of updated parameter sets, and where one or more of these parameter sets delays completion of the packet because of problematic solver convergence, PEST_HP can be instructed not to wait for such overdue model runs (if they are "overdue enough"). Instead, it abandons the troubled agents until they have completed these overdue runs (and ignores the results of those runs), while advancing to the next iteration of the inversion process wherein PEST_HP commences filling of a new Jacobian matrix. Furthermore, if the JACUPDATE control variable is set to greater than zero (so that Broyden Jacobian updates are undertaken during each iteration of the inversion process prior to initiating a second round of update-testing model runs), PEST_HP internally re-sets this control variable to zero. This action is undertaken under the premise that the run-time cost of testing a second set of parameter updates during each iteration of the inversion process is likely to be increased considerably by agents becoming bogged down with problematical model runs.

These options, together with a third option whereby a run agent can terminate a model run if it takes longer to execute than a pre-set time limit, are now discussed in greater detail.

7.2.2 The RUN_SLOW_FAC Variable

BEOPEST (and Parallel PEST) read the value of a variable named RUN_SLOW_FAC from the Parallel PEST run management file (named *case.rmf* where the PEST control file is named *case.pst*). However PEST_HP does not read a run management file; the value of RUN_SLOW_FAC can instead be supplied on the sixth line of the PEST control file. Recall from the PEST manual that RUN_SLOW_FAC is used to calculate the elapsed time at which a model run is deemed to be overdue; in assuming the worst (namely communications failure with the pertinent run agent), the run manager then reassigns the overdue model run to the fastest available idle agent. The elapsed time at which a model run is deemed to be overdue is calculated as RUN_SLOW_FAC times the anticipated model run time on the agent to which the model run is assigned.

As is discussed above, in contrast to BEOPEST, PEST_HP commissions an alternative agent to re-commence an overdue model run only where model runs are being undertaken for the purpose of finite-difference derivatives calculation. Where model runs are being undertaken in order to test parameter upgrades, PEST_HP handles overdue model runs in a different way; this is governed by the RUN_ABANDON_FAC variable discussed in the following subsection.

If a value for RUN_SLOW_FAC is not supplied by the user, PEST_HP uses a default value of 5.0. Use of such a high factor is based on the premise that in a cloud computing environment communication between the manager and its agents is unlikely to fail. Furthermore, all computers used for carrying out model runs are likely to have similar speeds. If, on a particular occasion, a model run is overdue, this is likely to be an outcome of the parameters with which the model was provided for that run. There is thus no need to undertake the same model run using an idle agent, as the outcome of that run is unlikely to change when using the new agent.

RUN_SLOW_FAC can be set to a different value if desired. To assign an alternative value to RUN_SLOW_FAC, add the string

```
run_slow_fac = value
```

to the sixth line of the PEST control file following the value of the NUMLAM (and optional JACUPDATE) variable. This is the same line of the PEST control file as that on which the strings “lamforgive” and “derforgive” can also be provided. (Note that, as stated above, “lamforgive” is ignored by PEST_HP, as its model run forgiveness behaviour when commissioning model runs to test parameter upgrades is governed by internal settings and/or by the RUN_ABANDON_FAC variable.) In the string “run_slow_fac = *value*”, *value* represents a number greater than 1.0. PEST_HP will accept a space on either side of the “=” symbol when reading this string; alternatively, spaces can be omitted. The string can be placed anywhere following NUMLAM (and optionally JACUPDATE), interchangeably with the “lamforgive” and “derforgive” strings. See appendix 1 for an example.

As stated, provision of a value for RUN_SLOW_FAC is optional.

BEOPEST (and Parallel PEST) have been modified to allow reading of RUN_SLOW_FAC where it is supplied in the manner described above. However for BEOPEST and Parallel PEST, a value supplied for this variable in the run management file over-rides a value supplied for this variable in the PEST control file.

Note that if PEST_HP’s first attempt to run the model at the very beginning of the inversion process leads to failure, PEST_HP will not repeat that run using a different agent. It assumes that a mistake has been made in model setup; accordingly, it terminates execution with an appropriate error message.

7.2.3 The RUN_ABANDON_FAC Variable

PEST_HP does not re-assign late model runs to other agents if a packet of model runs has been commissioned for the testing of parameter upgrades. As has already been discussed, the parameter upgrade testing process has been designed to finish quickly so that PEST_HP can move on to the next iteration of the inversion process with agent idleness minimized. However it is not impossible that certain updated parameter sets may precipitate interminably slow model execution speeds. An appropriate setting of the RUN_ABANDON_FAC variable can instruct PEST to abandon inordinately delayed model runs and move on to the next iteration of the inversion process. The RUN_ABANDON_FAC variable instructs PEST_HP to treat the agent as lost until the recalcitrant model run is complete. Once this occurs, the respective agent is then “brought back into the fold” and assigned a model run appropriate to PEST_HP’s current task. Meanwhile, as stated above, if the optional JACUPDATE variable is set to a number greater than zero, PEST_HP sets this variable to zero to reduce the chances of losing agents again.

The means through which a value is supplied for RUN_ABANDON_FAC are very similar to those for which a value is provided for RUN_SLOW_FAC. To assign a value to RUN_ABANDON_FAC, add the string

```
run_abandon_fac = value
```

to the sixth line of the PEST control file following the value of the NUMLAM (and optional JACUPDATE) variable; see appendix 1 for an example. The protocol is identical to that for RUN_SLOW_FAC. Note, however, that a RUN_ABANDON_FAC value of 0.0 deactivates run abandonment. (The same happens if no value is supplied for RUN_ABANDON_FAC at all.) If it is not assigned a value of 0.0, then RUN_ABANDON_FAC must be provided with a value of 1.2 or greater (preferably greater than 2.0); if not, PEST_HP will cease execution with an appropriate error message.

The way in which the RUN_ABANDON_FAC variable is employed in parallel run management is a little different from that in which the RUN_SLOW_FAC variable is employed Suppose that the slowest model run of the present package has taken *N* seconds to complete. RUN_ABANDON_FAC will abandon all non-completed runs after a further *N* × RUN_ABANDON_FAC seconds have elapsed if no further runs have reached completion in the meantime. However if, over this time, another model run reaches completion, then *N* is re-set and a new agent-abandonment threshold is calculated. (It is apparent from this strategy that tolerance for late model runs increases as PEST_HP discovers that lateness is the norm for the current parallel run package.)

7.2.4 The WIN_MRUN_HOURS Variable

In the Windows version of PEST_HP, a time limit can be placed on how long a model is allowed to run, regardless of the reason for carrying out this run. This variable is not available in the LINUX version of PEST_HP. In the LINUX environment, you can limit model execution times yourself using the *timeout* system command when running the model.

By way of example, suppose that you wish to terminate execution of the model if it runs for more than 30 minutes. (This can serve as a defence against excessive execution time caused by model solver convergence difficulties.) Then the following string should be added to the sixth line of the PEST_HP control file, anywhere following the value of the RLAMFAC variable.

```
win_mrun_hours = 0.5
```

This informs the PEST_HP run manager that it must instruct its agent to terminate a model run if that run takes longer than 0.5 hours. Once an agent has terminated execution of the model, it attempts to read the model’s output files in the usual manner. Naturally, these will be incomplete, so a run failure will be reported back to the run manager. The run manager then treats the expired run as it does any other failed model run.

Note the following.

- If the value of WIN_MRUN_HOURS recorded in the PEST control file read by the PEST_HP run manager differs from that recorded in a PEST control file read by an AGENT_HP run agent, the value read by PEST_HP takes precedence. Similarly, if a value for WIN_MRUN_HOURS is not supplied in the PEST control file that is read by an agent, but is recorded in that which is read by the PEST_HP run manager, then the PEST_HP run manager instructs its agents to terminate model runs once their allotted time has expired, and agents will obey.
- If the value supplied for WIN_MRUN_HOURS in the PEST control file read by the PEST_HP manager is less than or equal to zero, then model run termination functionality is disabled.
- If WIN_MRUN_HOURS is set to a positive number, then DERFORGIVE is automatically activated. That is, model run failure (or model run termination) is forgiven when calculating finite-difference derivatives.
- Model run termination acts independently of model run abandonment. A run can be abandoned by the PEST_HP run manager. However it will not be terminated by the AGENT_HP agent until the WIN_MRUN_HOURS model expiry time has elapsed. Run abandonment is done by the PEST_HP manager while run termination is a local affair, implemented by AGENT_HP using a run-time limit provided by the PEST_HP manager.

It is important to note that when WIN_MRUN_HOURS is set to a positive number, AGENT_HP uses the *CreateProcess()* function from the Windows API rather than a *system*() call to run the model. This function has certain idiosyncrasies. The following requirements of the command supplied in the "model command line" section of the PEST control file must be respected.

- Do not use the "<" character in the model command to direct the model to read keyboard input from a file. Similarly, do not use the ">" character to direct model output to a file (or to "nul"). If this is a requirement for running the model, place the command to run the model in a batch file, which is then run by the *CreateProcess()* function as the model command.
- If you run a batch file as the model command, be sure to include the ".bat" suffix in the command; the *CreateProcess()* function does not add this suffix itself.

7.3 Termination of PEST_HP

7.3.1 General

Reasons for graceful termination of PEST_HP execution include (but are not limited to) the following. (See the PEST manual for a description of pertinent variables.)

- NOPTMAX iterations have elapsed since execution of PEST_HP began.
- The (measurement) objective function is less than PHISTOPTHRESH.
- Parameter estimates have not improved over the last NPHISTP iterations. (Parameter "improvement" depends on whether PEST_HP is running in "estimation" or "regularisation" mode.)
- The measurement objective function is below PHIMACCEPT (unless REGCONTINUE is set to "continue".)
- The objective function has decreased by less than a relative amount of PHIREDSTP over NPHISTP successive iterations. (When run in "regularisation" mode the objective function in question may be the regularisation or measurement objective function, this depending on the value of the latter in relation to PHIMLIM.)

- The relative change in no parameter has exceeded RELPARSTP over NRELPAR successive iterations.
- The (measurement) objective function has fallen below PHISTOPTHRESH.

In many contexts of PEST_HP usage, it is better to stop PEST_HP yourself than to wait for one of these termination criteria to be met, especially if paying for time on the cloud. As the writer of PEST (and as its longest and most devoted user), I normally set termination criteria tightly in order to avoid premature cessation of PEST execution. However I monitor the progress of PEST myself. When it appears unlikely that further PEST execution will result in a further lowering of the objective function, I terminate it. (The easiest way to stop PEST_HP is to press <Ctl-C> when focussed on the PEST_HP manager window.) The exception to this rule may be when PEST_HP is running in "regularisation" mode and REGCONTINUE is set to "continue". In this case I am seeking the lowest regularisation objective function that I can, commensurate with PEST_HP achieving a measurement objective function of PHIMLIM. If the measurement objective function is below PHIMLIM, then I may allow PEST_HP to raise it to PHIMLIM while lowering the regularisation objective function a little further.

7.3.2 New Termination Criteria

PEST_HP accepts two variables, beyond those which are used by other versions of PEST, that control termination of its execution. These are the HARDSTOPHOURS and SOFTSTOPHOURS variables. Either of these variables can be used, or neither of them can be used. However they cannot be used together.

Suppose that SOFTSTOPTHOURS is provided with a value of 24.0. Then the PEST_HP manager will cease execution on the first occasion that it begins a new iteration of the inversion process following the passing of 24 hours since the commencement of its execution. Thus an upgraded parameter set is calculated before cessation of execution, and no model runs are wasted. When the PEST_HP manager then ceases execution, it signals all of its agents to also cease execution. Unless an agent is preoccupied in supervising an abandoned model run, it ceases execution immediately upon reception of this signal. In contrast, if it is undertaking an abandoned model run it cannot receive this signal until the run is complete; thereupon it ceases execution. Alternatively, it may terminate the model itself after WIN_MRUN_HOURS have elapsed.

If HARDSTOPTHOURS is set to 24.0, then the PEST_HP manager will cease execution 24 hours after commencement of its execution; regardless of the current status of the inverse problem solution process. There may be a slight delay however if parameter numbers are high and PEST_HP is engaged in laborious parameter update calculations. Upon cessation of its own execution, the PEST_HP manager signals to its agents to also cease their execution. However they will not receive this signal until the model runs which they are supervising are complete. (These can be stopped manually, of course, if desired.)

If PEST_HP ceases execution because of a SOFTSTOPHOURS or HARDSTOPHOURS timeout, then its execution can be easily resumed by restarting it using the "/s" switch.

7.3.3 Specifying Values for Timeout Variables

Values are provided for the SOFTSTOPHOURS or HARDSTOPHOURS variable by including the string

softstophours = *value*

or

```
hardstophours = value
```

anywhere on the ninth line of the PEST control file (the same line that is used by other PEST termination control variables); *value* must be replaced by a number specifying PEST_HP timeout time in hours. See appendix 1 for an example.

7.4 User-Prescribed Insensitivity

7.4.1 General

Like PEST and BEOPEST, PEST_HP supports the use of different model commands for running the model in order to calculate finite-difference derivatives with respect to different parameters. This occurs if the NUMCOM variable in the "control data" section of the PEST control file is set to a number greater than 1, and/or if PEST's observation re-referencing functionality is activated. In the former case, the command index that is associated with each parameter must be recorded in the "parameter data" section of the PEST control file; meanwhile the model commands themselves are listed in the "model command line" section of the PEST control file.

The use of different model commands to run a model for finite-difference calculation of sensitivities with respect to different parameters can be useful where models are complex, take a long time to run, and have many parameters. This functionality allows a modeller to forego the running of all components of a complex model when calculating derivatives for some parameters; for these parameters, only a shorter version of the model may need to be run when calculating sensitivities with respect to those parameters.

For example, suppose that a modeller wishes to calibrate a composite steady state and transient groundwater model. Suppose also that the model domain is populated with pilot points used for representation of both permeability and storage coefficient parameters. Suppose further that the transient component of the calibration process is focussed on a small part of the model domain in which one or a number of pumping tests was conducted, and that storage coefficient pilot points are restricted to this area. A modeller may set up two models as follows:

- model 1, in which both the steady state and transient components of the model are run; it is presumed that this model runs slowly because of its transient component.
- model 2, in which just the steady state component of the model is run; it is presumed that model 2 runs quickly.

It is important to note that when running model 2, the outputs of the transient component of the model must somehow be provided with values, as PEST expects to read all model output files using the instruction set which was constructed for this purpose. This same instruction set is used to read model-generated observations following all model runs.

The modeller may instruct PEST to run model 1 only when calculating finite-difference derivatives for storage coefficient parameters and for permeability parameters associated with pilot points which are situated close to the sites of the pumping tests. For all other parameters, model 2 is employed for finite-difference derivatives calculation. Let us refer to the first group of parameters as type 1 parameters and the second group of parameters as type 2 parameters.

A problem with this scheme in versions of PEST other than PEST_HP, is that the modeller must find a way of ensuring that sensitivities of transient model outputs with respect to type 2 parameters are zero. (Sensitivities of storage parameters to steady state model outputs are automatically zero.) One way to achieve this is to employ PEST's observation re-referencing functionality. The model run undertaken for re-referencing purposes would be comprised of both the steady state and transient models (i.e. model 1). Transient outputs of this re-referencing run would then be saved to a special file. Model 2 would copy the contents of this special file to the expected transient model output file on every occasion that it is run during the same iteration of the inversion process. These model outputs would therefore remain unchanged from their iteration-specific reference values as type 2 parameters are incrementally varied. Sensitivities of these outputs to type 2 parameters would therefore be zero.

While this scheme would be effective, it is somewhat cumbersome to implement. It becomes even more cumbersome to implement in a parallel environment where the transient output file produced during the re-referencing run must be copied to the working directories employed by all parallel run agents. (Note that PEST_HP's file distribution functionality could help here.)

"User-prescribed zero sensitivity" functionality implemented in PEST_HP can be of assistance in situations such as these. When using this option, there is no need to invoke PEST_HP's observation re-referencing functionality. (Note however that observation re-referencing is not precluded when deploying user-prescribed zero sensitivity functionality.) Instead, the modeller can program model 2 to endow all expected transient model outputs with a single, specific, value. (This can also be implemented using a "copy" command run through the model 2 batch file.) This single value (designated as ZEROSENVAL) instructs PEST_HP that the sensitivities of all model outputs which are assigned this value with respect to the parameter that is being incrementally varied are zero.

7.4.2 Implementing User-Prescribed Insensitivity

User-prescribed zero sensitivity functionality is implemented by adding a string such as the following to line 8 of the PEST control file, somewhere following the value of the PHIREDSWH variable (interspersed, if necessary, with other variables which can also be recorded on this line).

```
zerosenval = -1.11e29
```

This string instructs PEST_HP to interpret a model output value of -1.11E29 as a directive that it must award a sensitivity of zero to all model outputs that have this value to the parameter that is being incrementally varied on the pertinent model run. A modeller can select this value him/herself. However its absolute value must be less than 1E30. At the same time, he/she must ensure that pertinent model outputs have this exact value. In the above string, the spaces before and after the "=" symbol are optional.

If user-prescribed zero sensitivity functionality is operative, the user must ensure that a value of ZEROSENVAL (-1.11E29 in the above example) is not recorded by the model except when it is run for the purpose of finite-difference derivatives calculation. If PEST_HP detects this value on the initial model run, on a model run undertaken for observation re-referencing purposes, or on a run undertaken for testing parameter upgrades, it will cease execution with an appropriate error message.

7.5 Sensitivity Reuse

As is described in PEST documentation, PEST’s sensitivity re-use functionality can be activated by providing the string “senreuse” on the eighth line of the PEST control file. Optionally a “sensitivity reuse” section can also be provided in the PEST control file in order to assign values to control variables which govern the operation of PEST’s sensitivity reuse functionality. If this section is not supplied, then PEST provides default values for these control variables itself.

Experience demonstrates that reusing the sensitivities of relatively insensitive parameters, rather than re-calculating them during every iteration of the inversion process, can sometimes realize spectacular savings in model runs. However at other times the gains in overall model run efficiency are not so great, especially if model outputs used in the calibration process are highly nonlinear with respect to some or all of the model’s parameters; in cases such as these, the reduced cost per iteration achieved through sensitivity reuse may be outweighed by an increase in the number of iterations required to lower the objective function.

Where PEST_HP is deployed in a highly parallelized environment such as a computing cloud, then model run reductions should be achieved in packets that are equal to the number of available run agents at PEST_HP’s disposal (this is normally equal to the number of deployed cores). Ideally, a modeller should choose the number of agents available to PEST_HP such that the number of model runs required for filling of the Jacobian matrix is an integral multiple of these agents. Uncontrolled reductions in the number of model runs required for filling the Jacobian matrix precipitated by application of PEST’s sensitivity reuse functionality may result in some agents being idle for a while. It would be far better if these agents were calculating sensitivities, even if it transpires that some of these sensitivities exert little influence on ensuing parameter upgrade calculations.

During each iteration of the inversion process in which sensitivity reuse is possible, PEST_HP first identifies parameters which are candidates for sensitivity reuse in the same manner that PEST and BEOPEST identify these parameters, i.e. through their reduced composite sensitivities in comparison with other parameters. PEST_HP then counts the number of run agents that are currently at its disposal. If necessary, it then increases the number of parameters for which finite-difference calculation of derivatives will take place until the number of agents is an integral multiple of the number of model runs which will be carried out. In doing this, it accommodates the finite difference derivatives settings provided in the “parameter groups” section of the PEST control file, and whether or not the inversion process has reached the stage where higher order derivatives are being calculated instead of forward derivatives.

Suppose, for example, that 500 parameters are being estimated, and that PEST_HP is employing 50 agents. Suppose further that, during one particular iteration of the inversion process, sensitivity reuse functionality decrees 120 parameters to be insensitive enough to forgo finite-difference re-calculation of their derivatives during this iteration. If, at this stage of the inversion process, derivatives are being calculated using forward differences only, then PEST_HP will only reuse sensitivities for 100 parameters instead of 120 parameters in order to prevent 20 agents from being idle during part of the Jacobian matrix filling process. Of the 120 parameters for which it had previously decided that re-calculation of finite-difference derivatives should be foregone, it reinstates those which, according to previous iterations, are likely to be the most sensitive.

Note the following restrictions on the use of sensitivity reuse functionality (in all versions of PEST).

- A prematurely terminated PEST_HP run cannot be restarted using the “/s” switch if sensitivity reuse functionality is engaged. This is because sensitivities pertaining to the previous iteration of the inversion process are not recorded in the run restart file, as this would make that file unduly long.
- If sensitivity reuse functionality is activated, a covariance matrix cannot be supplied for any observation group that is not comprised entirely of prior information. This is because use of an observation covariance matrix can cause “crossover” in composite parameter sensitivities, possibly introducing sensitivity to otherwise insensitive parameters. In contrast, if usage of covariance matrices is restricted entirely to prior information equations, this poses no problems for sensitivity reuse. Sensitivities of prior information equations to parameters do not require calculation through finite differencing.

7.6 Suspension of Observation Re-referencing

Suppose that a modeller is using PEST_HP’s file distribution functionality for hastening execution speed of a steady state groundwater model or natural state geothermal reservoir model during calibration of this model. In this context, a user may choose to employ file distribution in conjunction with PEST_HP’s observation re-referencing functionality. Thus, after distribution of an initial state file, a special model run may be undertaken prior to running the model many times for filling of the Jacobian matrix. This special model run will be undertaken for the purpose of providing a set of reference observations based on newly upgraded parameters and the newly distributed initial state file, before these parameters are sequentially subject to incremental variation for the purpose of finite-difference derivatives calculation.

File re-distribution for this purpose is required at the beginning of each iteration of the inversion process except the first. Ideally, carrying out of the complementary observation re-referencing run should therefore be prevented during the first iteration of the inversion process. This run will not do any harm. However it takes up computing resources for no good reason (especially if the short steady state run is accompanied by an ensuing transient run).

Observation re-referencing can be prevented during the first iteration of the inversion process by placing the string “orr_not_first” on the fifth line of the PEST control file following the “obsreref” string. “Orr_not_first” stands for “observation re-referencing – not in first iteration”.

Note the following.

- If the “orr_not_first” string is supplied while the “obsreref” string is not supplied on the fifth line of the PEST control file, PEST_HP will cease execution with an appropriate error message.
- Because of the way in which it operates, observation re-referencing cannot be suspended during the first iteration of an inversion process in which more than one model command is employed for finite difference derivatives calculation with respect to different parameters. If an attempt is made to do this, PEST_HP will cease execution with an appropriate error message.

7.7 Alternative LSQR Settings

The roles of the LSQR_ATOL, LSQR_BTOL, LSQR_CONLIM and LSQR_ITNLIM variables are explained in PEST documentation. In general, the lower are the values ascribed to LSQR_ATOL and LSQR_BTOL, and the higher are the values ascribed to LSQR_CONLIM and LSQR_ITNLIM, the longer will the LSQR solution process take. A more precise solution to the equations which PEST uses to solve the linearized inverse problem can often be found with tighter settings for these variables.

Where parameters number in the thousands and observations number in the tens of thousands, the LSQR solution process may take a few minutes. (Nevertheless it is far faster than singular value decomposition). Unfortunately, if PEST_HP is run in "regularisation" mode (as it should be when estimating many parameters), the linearized inverse problem may need to be solved many times during each iteration of the overall inversion process in order to estimate the optimal regularisation weight factor for use during that iteration. This can add considerably to the overall length of the inversion process. To make matters worse, parallel run agents are standing idle while repeated solution of the linearized inverse problem is taking place to refine the regularisation weight factor.

This process can be hastened by using looser convergence criteria for calculation of the best regularisation weight factor to use during each iteration of the inversion process; see PEST documentation of the WFFAC and WFTOL regularisation control settings. Alternatively, or as well, PEST_HP can be asked to use looser LSQR settings when undertaking linearized testing of a regularisation weight factor than when actually solving for a parameter upgrade.

Figure 7.1 shows the "lsqr" section of a PEST control file. Values for LSQR_ATOL, LSQR_BTOL, LSQR_CONLIM and LSQR_ITNLIM are supplied on the third line of this section.

```
* lsqr
 1
 1e-4 1000 10000
 0
```

Figure 7.1 The "lsqr" section of a PEST control file.

Figure 7.2 shows this same section of the PEST control file in which alternative values are provided for LSQR control variables for use in calculating the regularisation weight factor. These alternative values must follow the string "alt_regwt". The presence of this string on this line of the PEST control file notifies PEST_HP that alternative values for LSQR control variables follow. (Note that, to avoid confusion, PEST_HP will not allow any text other than "alt_regwt" to follow the values of the normal LSQR control variables on this line of the PEST control file.)

```
* lsqr
 1
 1e-4 1000 10000 alt_regwt 1e-4 100 1000 4
 0
```

Figure 7.2 The "lsqr" section of a PEST control file in which alternative LSQR control variables are provided.

The final entry on the third line of the PEST control file fragment shown above is the value of the variable LSQR_STOPITER. This (integer) number specifies a PEST inversion iteration number. Once this iteration of the inversion process has been reached, PEST_HP no longer uses the alternative LSQR control variables when undertaking linearized testing of iteration-specific regularisation weight factors. Set this to a negative value or zero if you wish that PEST_HP never uses the alternative LSQR control variables for this purpose; set it to a very high number if you wish that PEST_HP always uses these variables for linearized regularisation weight factor testing. (Note that when the REGCONTINUE regularisation control variable is set to "continue", it is good to revert to the use of the tighter LSQR settings for linearized weight factor testing after a few iterations of the inversion process have elapsed.)

Another means through which the speed of the LSQR solution process can be considerably sped up is through use of the *pest_hp_mkl.exe* executable program in place of *pest_hp.exe*. See section 1.5 of this manual.

An alternative, much faster but approximate, means of calculating the regularisation weight factor is through use of the REG2MEASRAT regularisation control variable. This is now discussed.

7.8 High-Speed Regularisation

As is discussed in the previous section, when PEST is run in “regularisation” mode, it calculates a regularisation weight factor during every iteration of the inversion process. It uses a linear approximation to the inverse problem to evaluate a factor that will allow the measurement objective function achieved through the current iteration to approximately equal a user-supplied target measurement objective function; the latter is supplied as the PHIMLIM regularisation control variable. Unfortunately, as has been discussed, calculation of the regularisation weight factor in this fashion can be a numerically intensive procedure, especially where parameter and/or observation numbers are high. Additionally, this calculation can be unreliable when a Jacobian matrix is approximate (as occurs when it is calculated using random or simultaneous parameter increments; see sections 9 and 10 of this document).

A variable named REG2MEASRAT can be added to the first line of the “regularisation” section of a PEST control file. Provision of a non-zero value for this variable instructs PEST_HP to forego the laborious calculation of the regularisation weight factor in the manner described above, and to replace it with a far simpler means of calculating this weight factor. The user-supplied value of REG2MEASRAT must follow the string “reg2measrat=”; note that spaces can surround the “=” symbol if desired.

REG2MEASRAT must be greater than or equal to zero; it must also be less than 1.0. A value of zero disables REG2MEASRAT functionality so that the regularisation weight factor is calculated by PEST_HP in the usual manner. Provision of a non-zero value for REG2MEASRAT instructs PEST_HP to endow the regularisation weight factor employed for the current iteration with a value that ensures that the regularisation objective function is equal to REG2MEASRAT times the value of the measurement objective function at the beginning of the iteration.

Figure 7.3 shows the “regularisation” section of a PEST control file in which a non-zero value is provided for REG2MEASRAT. Omission of this variable from a PEST control file effectively endows it with a value of zero.

```
* regularisation
   0.1       0.105      .1   reg2measrat=0.1
 1.0   1.0e-10    1.0e10
 1.3   1.0e-2     1
```

Figure 7.3 “Regularisation” section of a PEST control file in which a value is supplied for the REG2MEASRAT control variable.

Use of the REG2MEASRAT control variable to govern calculation of the regularisation weight factor does not affect PEST_HP’s termination criteria when it is run in “regularisation” mode. In particular, PEST_HP will terminate execution if the measurement objective function falls below PHIMLIM (the target measurement objective function); hence the setting of PHIMLIM matters, even if it has no effect on the way in which the regularisation weight factor is calculated. Note also that “REGCONTINUE” is automatically set to “nocontinue”, regardless of its user-supplied setting, if REG2MEASRAT is set to a value greater than zero.

There is an idiosyncrasy associated with use of a non-zero value for REG2MEASRAT which deserves attention. Often the regularisation objective function is zero during the first iteration of an inversion process. This is certainly the case if the ADDREG1 or ADDREG2 utility is used to add regularisation to a PEST control file. Under these circumstances the regularisation objective function cannot be adjusted to be a user-defined fraction (i.e. REG2MEASRAT) of the current measurement objective function. Hence PEST_HP employs its usual algorithm for calculation of the regularization weight factor. This may be somewhat time-consuming; however its use is normally required only during the first iteration of the inversion process, and only under these special circumstances. To ensure that this process works, set the target measurement objective function (PHIMLIM), and the temporary target objective function (FRACPHIM) to values that you normally would when conducting PEST-based Tikhonov regularization.

7.9 Using the Marquardt Lambda for Regularisation

A variable named LAMBDA_FOR_REG can optionally appear on the first data line of the “regularisation” section of a PEST control file. See Figure 7.4. It can be assigned a value of zero or one. If omitted, its value is assumed to be zero. See Figure 7.4.

```
* regularisation
   0.1       0.105      .1    lambda_for_reg=1
 1.0   1.0e-10    1.0e10
 1.3   1.0e-2     1
```

Figure 7.4 “Regularisation” section of a PEST control file in which a value is supplied for the LAMBDA_FOR_REG control variable.

If LAMBDA_FOR_REG is set to 1, then the role of the Marquardt lambda changes. During each iteration of the inversion process, PEST_HP still tests different values of this variable, and undertakes model runs accordingly. Its value still defines an upgrade direction in parameter space. Model runs corresponding to fractional lengths in this direction can still be undertaken, especially if the number of agents to which PEST_HP has access is large, and/or if the UPTESTMIN control variable is set to a value such as 25 or 30 (see below), as it should be. However when LAMBDA_FOR_REG is set to 1, the value of the Marquardt lambda is actually the value of the regularisation weight factor. Therefore, the higher is the “Marquardt lambda” under these circumstances, the greater is the strength with which regularisation constraints imposed. Meanwhile, behind the scenes, PEST_HP employs a low value for true Marquardt lambda, increasing it, and maybe then decreasing it again, if the inversion process slows.

When LAMBDA_FOR_REG is set to 1, the trial-and-error strategy employed by PEST_HP for selection of “Marquardt lambda” values during any iteration of the inversion process is unchanged. That is, during any iteration of the inversion process, values that are employed for upgrade vector testing are centred on that which was most successful during the previous iteration. Fractional upgrade vector lengths are also chosen in the same way. However when PEST reports the value of the “Marquardt lambda” to the screen and to its run record file under these circumstances, it refers to it as the “reg factor”, just to make it clear that the role of this experimental variable has changed. However, it is still referred to as the “Marquardt lambda” in some ancillary files.

Ideally, with upgrade vector testing directions being set by experimental regularisation weight factors rather than by the actual Marquardt lambda, the latter should be set to a low value. As stated above, this is what PEST_HP does behind the scenes. At the end of each iteration, however, PEST_HP decides whether to raise or lower the value of the actual Marquardt lambda. It reports its choice to the screen and to the run record file. Sometimes it raises it considerably, on a temporary basis, in an attempt to hasten progress of a stalled inversion process.

If LAMBDA_FOR_REG is set to 1, then most other variables that appear in the “regularisation” section of a PEST control file (including REG2MEASRAT) become redundant, for most of these pertain to how the regularisation weight factor is back-calculated from a user-supplied target measurement objective function. Values for these variables must still be provided. However most of them are not reported in the run record file. (Only the IREGADJ regularisation control variable retains any meaning when LAMBDA_FOR_REG is set to 1.) Note, however, that PEST_HP will declare an inversion process to be over if the measurement objective functions falls below PHIMLIM.

As presently programmed, LAMBDA_FOR_REG can be set to 1 only if LSQR is used as a solution device for the inverse problem; thus LSQRMODE must also be set to 1.

Experience shows that setting LAMBDA_FOR_REG to 1 can sometimes assist the progress of an inversion process when Jacobian matrices are calculated using random increments of individual parameters. See the pertinent section of this manual for a description of PEST_HP’s random Jacobian functionality.

7.10 Marquardt Lambdas for SVDMODE Equal to 2

7.10.1 Calculating Parameter Upgrades

When the SVDMODE variable in the PEST control file is set to 2, PEST undertakes singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ where $\mathbf{J}$ is the Jacobian matrix and $\mathbf{Q}$ is the weight matrix. $\mathbf{Q}$ is often diagonal, with elements equal to the squares of user-supplied weights. However $\mathbf{J}$ can also include Tikhonov regularisation which may be accompanied by one or more covariance matrices. In that case $\mathbf{Q}$ includes the inverse of these matrices as well.

With SVDMODE set to 1, PEST computes parameter upgrades $\delta\mathbf{k}$ using the equation

$$\delta\mathbf{k} = \mathbf{V}_1\mathbf{S}^{-2}{}_1\mathbf{V}^{t}{}_1\mathbf{Z}^{t}\mathbf{Q}\mathbf{h} \tag{7.1}$$

where $\mathbf{V}$ and $\mathbf{S}$ are obtained through singular value decomposition of $(\mathbf{J}^{t}\mathbf{Q}\mathbf{J} + \lambda\mathbf{I})$ as

$$(\mathbf{J}^{t}\mathbf{Q}\mathbf{J} + \lambda\mathbf{I}) = \mathbf{V}\mathbf{S}^{2}\mathbf{V}^{t} \tag{7.2}$$

and λ is the value of the Marquardt lambda. The “1” subscript on $\mathbf{V}$ and $\mathbf{S}$ in equation 7.1 signifies truncation at an appropriate singular value index. Normally the truncation point is chosen using the EIGTHRESH variable provided in the “singular value decomposition” section of the PEST control file. If so, truncation is such that the ratio of lowest to highest retained singular value squared (i.e. the ratio of the lowest to highest diagonal elements of $\mathbf{S}^{2}$) is no smaller than EIGTHRESH. A suitable value for EIGTHRESH is about 5×10^{-7}. This is the value which prevents numerical noise incurred through finite difference calculation of elements of the Jacobian matrix from unduly contaminating values estimated for parameters.

7.10.2 Marquardt Lambda Values

Implementation of Marquardt lambda functionality is different when SVDMODE is set to 2. With SVDMODE set to 2, PEST computes a parameter upgrade using the equation

$$\delta\mathbf{k} = \mathbf{V}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t{}_1\mathbf{Q}^{½}\mathbf{h} \tag{7.3}$$

where, as stated above, $\mathbf{V}_1$, $\mathbf{S}_1$ and $\mathbf{U}^t{}_1$ are derived from singular value decomposition of $\mathbf{Q}^{½}\mathbf{J}$. The Marquardt lambda is handled in the following manner.

Let the current value of the Marquardt lambda be signified as λ. A number λ_a is added to all diagonal elements of $\mathbf{S}^{-2}$. λ_a is calculated from λ using the formula

$$\lambda_a = \lambda \times \mathbf{S}^{-2}{}_{(1,1)} \times \text{EIGTHRESH} \tag{7.4}$$

In equation 7.4 $\mathbf{S}^{-2}{}_{(1,1)}$ is the first diagonal element of $\mathbf{S}^{-2}$. That is, it is the squared inverse of the first singular value. If the current value of λ is above 1.0, the addition of λ_a to all diagonal elements of $\mathbf{S}^{-2}$ ensures that the truncation point moves far to the right, perhaps so far to the right that there is no truncation at all. If λ is high enough, δ**k** is effectively calculated using the method of steepest descent. On the other hand, as λ falls below 1.0, the truncation point is shifted dramatically to the left. If λ is low enough, the truncation point is determined solely by EIGTHRESH.

It should be noted that where Tikhonov regularisation is employed in solution of an ill-posed inverse problem, singular values do not fall to zero (if Tikhonov regularisation achieves the purpose for which it is intended). It should also be noted that, as the inversion process progresses, PEST_HP centres its selection of Marquardt lambdas on those which it believes from experience in calculating upgrade vectors so far, will be most effective in achieving maximum reduction of the objective function.

Limited experience to date suggests that the above methodology for deployment of the Marquardt lambda when SVDMODE is set to 2 works well when using a randomized Jacobian matrix (see section 9), and when solving highly nonlinear, highly ill-posed inverse problems.

7.11 Switching to Higher Order Derivatives

If the FORCEN control variable assigned to any parameter group in the “parameter groups” section of a PEST control file is set to “switch”, then PEST will switch from two point derivatives to higher order derivatives if, on any iteration of the inversion process, the objective function fails to improve by a relative amount of more than PHIREDSWH. However, the iteration at which the switch occurs can be postponed using the NOPTSWITCH variable. This prevents wastage of model runs where the objective function was not lowered as much as it otherwise would have been if the parameter upgrade vector had not been shortened through a parameter encountering its RELPARMAX, FACPARMAX or ABSPARMAX(N) change limit. (Note that a rise in the objective function will always precipitate the switch to higher order derivatives, regardless of the NOPTSWITCH setting.)

The NOPTSWITCH variable is optional. If you do not supply a value for this variable in the PEST control file, then PEST will switch to higher order derivatives calculation as early as at the end of the first iteration of the inversion process if the relative objective function reduction achieved during that iteration is less than PHIREDSWH. Unfortunately, this is not an uncommon occurrence. Furthermore, a limited fall in the objective function during the first iteration does not necessarily signify the need for more precise derivatives. More often than not, it reflects the fact that some parameters need to change a long way from their initial values in order to have an effect on the objective function, but that the length of the parameter upgrade vector was limited by RELPARMAX, FACPARMAX or ABSPARMAX(N).

PEST_HP has been programmed to recognize this situation. If the user does NOT supply a value for NOPTSWITCH (which is most often the case), then PEST will not switch to higher order derivatives at the end of the first or second iteration of the inversion process if any parameter encounters its RELPARMAX, FACPARMAX or ABSPARMAX(N) change limit during these iterations, as long as there was at least some improvement in the objective function. There are many occasions where preventing the premature onset of higher order derivatives calculation in this fashion can provide efficiency benefits that extend beyond these initial iterations. It is often in the first and second iterations that parameters need to change the most, and are hence most likely to encounter parameter change limits. On subsequent iterations they may not need to change as much in order to support a considerable lowering of the objective function. PEST_HP may therefore not need to switch to higher order derivatives until much later in the inversion process.

Note however that if NOPTSWITCH receives a value in the PEST control file, this value will be respected by PEST_HP; it will not switch to higher order derivatives calculation until the end of the NOPTSWITCH-nominated iteration unless the objective function fails to fall at all during a particular iteration. Note also that if NOPTSWITCH is set to 1 or 2, then PEST_HP will not over-ride the onset of higher order derivatives calculation at the end of these iterations if the objective function fails to fall by more than PHIREDSWH during these iterations because of change-limiting of the parameter upgrade vector.

7.12 BOUNDSCALE and JACUPDATE

Versions of PEST other than PEST_HP disable Broyden Jacobian updating if the BOUNDSCALE variable is set to “boundscale” in the “control data” section of the PEST control file. As is described in PEST documentation, the scaling of parameters according to their respective bounds intervals serves a number of useful purposes. These include a higher likelihood of obtaining estimates of parameters which are of minimized error variance, and a higher likelihood of calculating those estimates with numerical stability.

The PEST_HP inversion algorithm has been amended such that incompatibilities between Broyden Jacobian updating and parameter bounds interval scaling have been removed. Note however that, as for other versions of PEST, parameter bounds interval scaling cannot be implemented unless singular value decomposition or LSQR is used as a solution device for the inverse problem (which is recommended practice).

7.13 The UPTESTMIN and UPTESTLIM Variables

See sections 2.2 and 2.3 of this document for a description of the role of the UPTESTMIN and UPTESTLIM variables in limiting the number of model runs employed for testing parameter upgrades.

7.14 Model Run Failure

When using PEST_HP in conjunction with a complex model, there can be occasions where the model fails to run properly when provided with a certain set of parameters. Despite gross errors in its calculations, the model may nevertheless write its expected output files; however the numbers which it records on these files may be nonsense.

If this occurs during testing of parameter upgrades, its occurrence is normally indicated by a very high objective function. This does not trouble PEST_HP. It simply ignores the run; hopefully a more reasonable objective function can be achieved with an alternative set of parameters. However if this occurs during filling of the Jacobian matrix in response to incremental variation of a parameter, then the outcome will probably be an abnormally high set of sensitivities of model outcomes with respect to that parameter. This normally renders calculation of a useful parameter upgrade vector impossible, as the Jacobian matrix possesses one or a number of singular values which are very much higher than the others. PEST_HP then attempts to adjust the parameters that resulted in model run failure, while ignoring all other parameters. At best, this can lead to no improvement in the objective function; at worst it can promulgate a significant deterioration in the objective function. Experience has demonstrated that, under some circumstances, it can even result in failure of the algorithm that calculates parameter upgrade vectors (this is from the LAPACK family), whereby it gets lost in an infinite loop. Obviously, this situation must be avoided at all costs.

PEST_HP provides two variables that have been designed to detect and/or alleviate this situation. These variables are named JCOWARNTHRESH and JCOWARNZERO. Both of these should be set to a suitably high value; a value of at least 1E20 is suggested.

Suppose that JCOWARNTHRESH is set to 1E20. Then, immediately after filling the Jacobian matrix, PEST_HP warns the user (on its screen and on its run record file) if the absolute values of any Jacobian matrix elements are greater than 1E20. It will, in fact, count the number of elements whose values exceed this threshold and report this number.

Suppose that JCOZEROTHRESH is set to 1E20. Then not only will PEST_HP count the number of Jacobian matrix elements whose absolute values are greater than this threshold and report this to the screen and to its run record file; it will also alter the values of these elements to zero. Thus it may be possible for an unattended PEST_HP run to make progress, despite intermittent model run failure when filling the Jacobian matrix.

Values for JCOWARNTHRESH and JCOZEROTHRESH must be placed on the eighth line of the PEST control file. This is the line that begins with the value of the PHIREDSWH control variable. Figure 7.5 shows an example.

```
* control data
restart estimation
     104      14      13       0       6
     1     2 single  point  2   0   0
 10.0  -3.0  0.3  0.03  10 999  uptestmin=15
 10.0  10.0  0.001
 0.1  noaui jcowarnthresh=1.0E10 jcozerothresh=1.0E30
 30  0.005  4  4  0.005  4
 1  1  1
```

Figure 7.5 "Control data" section of a PEST control file in which values are supplied for the JCOWARNTHRESH and JCOZEROTHRESH control variables.

Note the following usage details.

- PEST_HP will allow values to be supplied for one or both of JCOWARNTHRESH and JCOZEROTHRESH.
- Values supplied for each of these variables must be zero or greater. A value of zero disables the variable.
- The default value for JCOZEROTHRESH is 0.0. The default value for JCOWARNTHRESH is 1e30.
- If positive values are supplied for both JCOWARNTHRESH and JCOZEROTHRESH, PEST_HP insists that the value supplied for JCOZEROTHRESH exceed that supplied for JCOWARNTHRESH.
- PEST_HP does not check prior information sensitivities against the JCOWARNTHRESH and JCOZEROTHRESH thresholds, as these are supplied directly by the user.

7.15 Observation Penalties

PEST_HP (and other members of the HP suite such as CMAES_HP, RSI_HP and JACTEST_HP) can accommodate "one way observations". Observations of this type must be assigned to observation groups whose names begin with "<@" or ">@". The first character of this unusual character sequence depicts the direction of the penalty. The second character (i.e. "@") attempts to ensure that such a group will not be named accidently by a user who is unaware of the treatment which PEST_HP gives to observations which belong to these groups.

Suppose that an observation belongs to an observation group whose name begins with ">@". Then if the model-calculated counterpart to this observation is less than its observed value, PEST_HP will adjust this value upwards until it equals the observed value. The residual thus becomes zero. Furthermore, the objective function is not increased under these circumstances. Conversely, if the model-calculated value of the observation is above the observed value, then the residual is calculated in the normal way, and the objective function suffers a corresponding increase. The ">" part of the observation group name thus denotes the direction from the observed value for which a penalty is incurred for model-to-measurement discrepancy.

The opposite applies to an observation that belongs to an observation group whose name begins with "<@". In this case an objective function penalty is incurred only if the model-generated counterpart to the observation is less than the observed value. If it is above the observed value, the model-generated counterpart to the observation is set to the observed value, so that the residual is zero.

This implementation of one-way penalties requires minimum alteration to PEST_HP source code. However, it may cause some confusion when a user inspects model outputs as recorded on the run record file (*case.rec*) and in residuals files (*case.res* and *case.rei*). Model outputs that are above or below observed values, and that belong to observation groups whose names begin with "<@" and ">@" respectively, will be recorded as being equal to those values. This, of course, is incorrect. On being surprised, the user must remember what he/she is reading right now.

Similarly, a model output that is above or below the observed value of a field observation will have a sensitivity of zero to all model parameters if it belongs to an observation group whose name begins with “<@” or “>@” respectively.

At the time of writing, the non-HP version of PEST does not include this functionality. Hence if a PEST control file includes one or more observation groups whose names begin with “<@” or “>@”, the objective function calculated by PEST will be different from that calculated by PEST_HP.

In order to conform a little more closely with the PEST++ protocol for one-way penalty functions, members of the PEST_HP suite also permit the use of “<~” and “>~” as observation group name prefixes. Use of these strings for commencement of the name of an observation group has exactly the opposite effects to use of “<@” and “>@”.

8. Stopping and Re-Starting PEST_HP

8.1 Resumption of Execution

Execution of PEST_HP can be terminated abruptly by typing <Ctl-C> while focussed on its working window. Alternatively, its execution can be terminated in a gentler manner using the PSTOP or PSTOPST commands; see below. Inadvertent termination of PEST_HP execution can follow power or network failure.

If execution of PEST_HP is prematurely terminated in any of these ways, it can be restarted using the “/s” switch. If its execution was terminated while undertaking model runs required for filling of the Jacobian matrix (normally the most time-consuming part of the inversion process), then PEST_HP will re-commence execution at exactly the same location in its processing sequence as that at which its execution was previously terminated. It will read the outcomes of model runs previously undertaken for filling of the Jacobian matrix from files that it uses for parallel run management; it will then complete the filling of this matrix by instructing its agents to undertake the remaining model runs.

If execution of PEST_HP was terminated while model runs were being undertaken for the purpose of testing parameter upgrades, PEST_HP will re-commence execution at that point of its processing sequence where upgrade calculation commenced. Normally, this does not constitute a degradation of efficiency because (as is described in section 2 of this document) PEST_HP often undertakes only one set of parallel runs during that phase of the inversion process in which upgraded parameter sets were calculated and tested. If PEST_HP execution was inadvertently terminated while these runs are being undertaken, they would need to be re-initiated anyway. Nevertheless, if model run times are significantly different for different parameter sets, there will be some wastage of model runs if PEST_HP execution was terminated after some model runs had been completed but before others had been completed. However this potential for run loss must be balanced against the fact that a restarted PEST_HP may not have the same number of agents at its disposal as the previously terminated PEST_HP. The Marquardt lambda selection strategy may therefore be different between the old and new PEST_HP runs.

If Broyden Jacobian updating is activated, then restart functionality presently implemented by PEST_HP will indeed result in the need to repeat the set of model runs based on the unimproved Jacobian matrix if cessation of execution occurred while model runs were being undertaken based on parameter upgrades calculated using the Broyden-improved Jacobian matrix.

If execution of PEST_HP was inadvertently terminated after commencement of execution using the “/f” command line switch, then its execution can be resumed using the “/s” switch, just as for a PEST_HP run that was undertaken for the purpose of solving an inverse problem. In this case execution of the re-started run should also be initiated using the “/f” switch. For example, if the PEST_HP manager was originally started using the command

```
pest_hp case /f /h :4004
```

then an interrupted run should be restarted using the command

```
pest_hp case /f /s /h :4004
```

If restarted in this manner, PEST_HP does not prompt for the names of the parameter value files which it must read, nor for the size of a parallel run packet, as it did on its original run. Instead, it obtains this information from its restart file; it then recommences its run at that point in its processing sequence at which its execution was previously terminated. Information calculated during the resumed run is appended to the original run record and run results files (as if execution of PEST_HP had never been interrupted in the first place).

8.2 Stopping and Pausing

In common with normal PEST and BEOPEST behaviour, the typing of “pstop” or “pstopst” in another command line window open to the PEST_HP manager’s folder instigates cessation of PEST_HP execution; run agents cease execution when the model runs which they are respectively supervising are complete. If stopped using the “pstop” command, cessation of execution of the PEST_HP manager is immediate. If stopped using the “pstopst” command, the manager records information pertinent to the current inversion process at the end of its run record file before ceasing execution. In either case, execution of the prematurely terminated PEST_HP run can be resumed using the “/s” command line switch as discussed above.

As for the normal PEST and BEOPEST, execution of PEST_HP can be paused and resumed using the “pause” and “unpause” commands. As for the “pstop” and “pstopst” commands, these commands must be issued from a command line window which is open to the PEST_HP manager’s working folder.

8.3 Special Considerations for the “/f” Switch

If PEST_HP is run using the “/f” switch, so that it records the outcomes of a sequence of model runs in a run results file, then its execution can be terminated using either the “pstop” or “pstopst” commands in the manner described above. However the PEST_HP manager responds slightly differently to each of these two commands. If stopped using the “pstop” command, cessation of the PEST_HP manager’s execution is immediate. However if stopped using the “pstopst” command, the manager empties the parallel run register before ceasing execution. Thus model output values corresponding to model runs comprising the latest run packet which have already been completed are recorded in the run results file; meanwhile, corresponding objective functions are recorded in the run record file and written to the screen.

If stopped using the “pstop” or “pstopst” command, PEST_HP execution can be resumed using the “/s” switch together with the “/f” switch in the manner described above. However if execution of PEST_HP was terminated using the “pstopst” command, some information will be duplicated in both the run record and run results files upon resumption of PEST_HP execution, namely the outcomes of model runs comprising part of an interrupted run package which were actually completed prior to termination of the previous PEST_HP run. As stated above, these model run outcomes are recorded in these PEST_HP output files in accordance with its programmed response to the “pstopst” command.

9. Randomized Jacobian

9.1 Introduction

This section describes functionality included in PEST_HP that often allows progress to be made with a high level of model run efficiency in difficult parameter estimation contexts, especially those in which parameter numbers are large and where the dimensionality of the calibration solution space is small. This progress may come at a price, however. While the methodology described herein may allow a good fit to be attained between model outputs and a calibration dataset, the solution to the inverse problem that is thereby achieved may not be that of minimized error variance. However, if proper precautions are taken, the solution achieved through the methodology described herein may not deviate too far from this optimal solution.

Deviations from inverse solution optimality can arise where parameter upgrade directions are not confined to the calibration solution space. Consequently, null space components are introduced to parameters as they are upgraded. These are not desirable features of the so-called "calibrated parameter field". However this is the price to be paid for the speed with which a good fit can be attained with a calibration dataset where parameter upgrades are based on a randomized Jacobian matrix. Furthermore, if steps are taken to keep such "null space entrainment" to a minimum, the calibrated parameter field may still be quite acceptable.

Because of its model run efficiency, the methodology described herein may greatly reduce the numerical burden of direct predictive hypothesis testing, whereby an hypothesized prediction is included in the calibration dataset. Model run efficiency is particularly important under these circumstances as a model must simulate both past and future system states.

9.2 Overview

In its normal operation, PEST_HP approximates differentials comprising a Jacobian matrix using finite parameter differences. Under these circumstances, filling of the Jacobian matrix requires at least as many model runs as there are adjustable parameters. Where an inverse problem is ill-posed, the Jacobian matrix can then be processed using a subspace method such as singular value decomposition (SVD). SVD reduces the dimensionality of an inverse problem by selecting combinations of parameters for estimation in place of individual parameters. In doing this, it notionally formulates a full-rank Jacobian matrix from the original, rank-deficient Jacobian matrix; this full rank Jacobian matrix complements the solution subspace.

Methods such as randomized SVD have received a considerable amount of attention in the recent numerical methods literature; see, for example, Halko et al (2011) for a review. They are closely related to the use of ensemble methods as a solution device for ill-posed inverse problems. Ensemble methods have been used for both calibration and calibration-constrained uncertainty analysis; see, for example, White (2018) and Chada et al (2018). The utility of methods based on random realizations of parameters rests on the premise that a Jacobian matrix that is employed in solution of an ill-posed inverse problem need only have a rank that is as large as that of the matrix which SVD will ultimately derive to estimate values for a limited number of estimable parameter combinations. The rank of this matrix is equal to the number of parameter combinations which are thus estimated (i.e. to the dimensionality of the calibration solution space). Construction of a

Jacobian matrix which is rank-sufficient in terms of the dimensionality of the calibration solution space, while being rank deficient in terms of the number of adjustable parameters, can be achieved if random combinations of parameters are employed to explore the range space of this matrix. The dimensionality of its range space is equal to the number of parameter combinations which can be estimated, and hence to the dimensionality of the solution space. The number of sampled parameter sets that are required to explore this space and define its dimensions need only be slightly greater than the dimensionality of the solution space.

When employing PEST_HP's randomized Jacobian functionality, random parameter sets can be generated by PEST_HP itself. Alternatively, they can be provided by the user. Both of these means of random parameter set generation can be combined in a single PEST_HP run. If random parameter sets are generated by PEST_HP, every parameter is considered to be statistically independent of every other parameter; the characteristics of random parameter sets are inherited from the variables which would otherwise govern parameter variation for the purpose of finite-difference derivatives calculation. Alternatively, user-supplied random parameter sets can be generated using whatever specifications of parameter randomness that a user sees fit. Furthermore, PEST_HP can alter its source of random parameter sets from iteration to iteration of the inversion process – either randomly, or according to a user-specified strategy.

Regardless of the source of random parameter vectors, the deleterious effects of using a rank-deficient Jacobian matrix can be mitigated through deployment of an appropriate "localization" strategy. This can dampen the occurrence of phantom parameter-to-observation sensitivities arising from the use of an insufficient number of random parameter increments. Localization can be implemented automatically by PEST_HP; alternatively, or as well, a user can implement it him/herself by supplying his/her own localization matrix.

As stated above, the number of random parameter sets for which model runs are carried out in order to construct a Jacobian matrix of sufficient rank to allow model outputs to fit a calibration dataset can be considerably less than the actual number of parameters that are awarded to a model. Naturally, PEST_HP parallelizes these model runs. Once a rank-deficient approximation to a Jacobian matrix has been calculated using these random parameter set realizations, solution of the inverse problem can then be undertaken using any of the methods normally supported by PEST_HP. Singular value decomposition with SVDMODE set to 2 is often recommended as PEST_HP's use of different Marquardt lambdas when testing parameter upgrades is then able to achieve singular value compression, the benefits of which are described by Engel et al (2017).

As usual, it is recommended that the inverse problem solution process includes Tikhonov regularisation as a device for maximizing parameter reasonableness. As is discussed elsewhere in this manual, the numerical burden of Tikhonov regularization can be reduced considerably through selection of the "high-speed regularization" option accessed through the REG2MEASRAT control variable in the "regularisation" section of a PEST control file.

9.3 Randomized Jacobian Matrix: Theory

9.3.1 Calculating the Jacobian Matrix

If a model is linear, its operation can be represented by a matrix. Let this matrix be designated as **J** when the model is run under calibration conditions. Let us represent model outputs that correspond to members of a calibration dataset by the vector **o**. Let the vector **k** represent model parameters. Then

$$\mathbf{o} = \mathbf{Jk} \tag{9.1}$$

Suppose that elements of **k** are randomly generated based on a covariance matrix D(**k**) (which we distinguish from C(**k**), which is often used in PEST and related documentation to denote the prior parameter covariance matrix). Then the joint covariance matrix of **o** and **k** can be computed as

$$\mathrm{C}\left(\begin{bmatrix}\mathbf{o}\\\mathbf{k}\end{bmatrix}\right) = \begin{bmatrix}\mathbf{J}\\\mathbf{I}\end{bmatrix}\mathrm{D}(\mathbf{k})\begin{bmatrix}\mathbf{J}^{\mathrm{t}} & \mathbf{I}\end{bmatrix} \tag{9.2}$$

After carrying out the matrix products implied by equation 9.2, and extracting the pertinent submatrix from the resulting full matrix, we find

$$\mathrm{C}(\mathbf{o},\mathbf{k}) = \mathbf{J}\mathrm{D}(\mathbf{k}) \tag{9.3}$$

where C(**o**,**k**) expresses covariances between the elements of **o** and those of **k**. From (9.3),

$$\mathbf{J} = \mathrm{C}(\mathbf{o},\mathbf{k})\mathrm{D}^{-1}(\mathbf{k}) \tag{9.4}$$

For a nonlinear model, the Jacobian matrix **J** constitutes a local linearization of the action of the model on the set of parameters achieved at a certain stage of the inversion process; we denote this set of parameters as $\underline{\mathbf{k}}$.

An empirical estimate of C(**o**,**k**) can be obtained by running the model using different random realizations of **k** centred on $\underline{\mathbf{k}}$. The covariance between the *i*'th element of **o**, o_i, and the *j*'th element of **k**, k_j, can then be calculated as

$$\mathrm{C}(\mathbf{o},\mathbf{k})_{i,j} = \frac{\Sigma(o_i - \underline{o_i})(k_j - \underline{k_j})}{N-1} \tag{9.5}$$

where the summation is undertaken across all random parameter sets. In equation 9.5 $\underline{k_j}$ is the mean value of parameter k_j over all realizations while $\underline{o_i}$ is the mean value of model output o_i over all realizations.

For random parameter increments generated by PEST_HP, D(**k**) is a diagonal matrix. Each element of the diagonal is the variance (square of standard deviation) of the respective, locally randomized, parameter. This is because, in calculation of the covariance matrix featured in equation 9.3, PEST_HP considers individual parameters comprising elements of the vector **k** to be statistically independent of each other. Furthermore, parameter standard deviations are generally small. (However larger standard deviations can be employed if desired.) These will generally be of similar magnitude to the increments that would otherwise be used for calculation of a Jacobian matrix using finite parameter differences. The use of small standard deviations serves two purposes.

1. Elements of the matrix **J** (the local Jacobian matrix) calculated using equation 9.4 are a better approximation to true derivatives than if parameter randomization was based on larger standard deviations; hopefully, this accelerates solution convergence of the inverse problem.
2. For some models (for example geothermal reservoir models run under natural state conditions, and groundwater models run under steady state conditions), execution speed

can be dramatically increased if initial system states supplied to the simulator's solver approximate the actual simulated system states. PEST_HP can provide these initial states to the model from previously-calculated solution states, and update them during every iteration of the inversion process, using its file distribution functionality. However system state solutions determined during a previous iteration of the inversion process can only serve as useful initial system state conditions for the next iteration of that process if random parameter variations from their current reference values are restricted in size.

Where random parameter increments are provided by a user, D(**k**) may not be diagonal. For example, random parameter increments may have been generated using a covariance matrix that is proportional to the prior parameter covariance matrix, C(**k**). This is the user's choice. PEST_HP does, however, expect that random parameter increments have a mean of zero for non-log-transformed parameters and a mean of one for log-transformed parameters. Randomized increments are added to current parameter values in the former case and multiply current parameter values in the latter case. (If random parameter increment sets are generated using the RANDPAR4 utility supplied with the PEST suite, this will be achieved automatically.)

Let the matrix **K** denote a user-supplied parameter increment ensemble. This matrix has as many rows as there are adjustable parameters and as many columns as there are parameter increment realizations. PEST_HP calculates an approximate covariance matrix D(**k**) from **K** as

$$\mathrm{D}(\mathbf{k}) = \frac{\mathbf{K}^{\mathrm{t}}\mathbf{K}}{N-1} \tag{9.6}$$

Because D(**k**) is rank-deficient if the number of parameters is greater than the number of parameter increment realizations (which is generally the case), it cannot be inverted in accordance with the requirements of equation 9.4. Furthermore, if it were of full rank (this requiring at least as many parameter increment realizations as there are adjustable parameters), inversion of this matrix would incur a high numerical burden in highly-parameterized settings. PEST_HP overcomes the first of these problems by first subjecting the matrix **K** to singular value decomposition as

$$\mathbf{K} = \mathbf{USV}^{\mathrm{t}} \tag{9.7}$$

and then calculating the pseudoinverse $\mathbf{K}^{\mathrm{t}}\mathbf{K}^{-}$ of $\mathbf{K}^{\mathrm{t}}\mathbf{K}$ as

$$\mathbf{K}^{\mathrm{t}}\mathbf{K}^{-} = \mathbf{US}^{-2}\mathbf{U}^{\mathrm{t}} \tag{9.8}$$

This methodology for calculating an approximation to **J** is not unlike that described by Chen and Oliver (2013) to support their Jacobian-enhanced, ensemble Kalman smoother scheme. However in their case parameter randomization plays two roles. Firstly, it is employed to create parameter fields which constitute an evolving ensemble which is used to quantify parameter and predictive uncertainty. Secondly, the outcomes of model runs conducted using these parameter fields are employed in equations that are not unlike those discussed above to compute an approximation to the Jacobian matrix on which modification of these parameter fields is based. An important difference between the Chen and Oliver randomization scheme and that described in the present document is that parameter variances employed in the present scheme can (and probably should) be much smaller than those used by Chen and Oliver. This is because, in the present case, randomness applies to parameter increments rather than to parameters themselves. These increments are generated for use in model runs that are dedicated solely to construction of an approximate Jacobian matrix. The latter is then used to adjust a single parameter field to better satisfy calibration constraints. Another important difference is that, with appropriate selection of pertinent control variables, the rank-deficiency of the Jacobian matrix computed by PEST_HP decreases with progress of the inversion process. The Jacobian matrix that is achieved at the end of this process may therefore constitute a reasonable approximation to the true parameter Jacobian matrix. As such, it may form the basis for approximate linear parameter uncertainty and identifiability analysis.

As has already been mentioned, the method described herein also bears some resemblance to randomization algorithms which are used for singular value decomposition of large matrices. A major difference between these methods and that described herein however is that randomization is used to build an approximation to the Jacobian matrix herein, rather than to decompose an existing Jacobian matrix. However if the model to which PEST_HP is linked possesses an adjoint solver, so that "backward" model runs can be undertaken to directly compute matrix multiplications involving $\mathbf{J}^t$, the distinction between these two roles can fade. Algorithms described by Halko et al (2011) and Tropp et al (2016) could then be employed for rapid calculation of upgrade vectors without the subspace misalignment problems that attend the present methodology.

9.3.2 Localization

"Localization" is a term borrowed from ensemble Kalman filter and smoother terminology. It describes a process through which errors in a Jacobian matrix incurred through the use of random parameter increments are mitigated. At the same time, the degree of rank-deficiency of the Jacobian matrix can be reduced. Localization reduces the values of randomized Jacobian matrix elements (possibly to zero) if it is possible that these values may, in fact, be random noise incurred through the randomized Jacobian matrix construction process. This strategy is referred to as "localization" because sometimes a modeller may know that one parameter cannot influence a particular model output because the latter is far removed in space from the former; in this case he/she is entitled to replace a spurious sensitivity with the known value of zero.

PEST_HP allows a user to supply a "localization matrix". Let us refer to this matrix as **L**. It must have the same dimensions as the Jacobian matrix **J**. Then, on each occasion that PEST_HP calculates a Jacobian matrix using equation 9.4, it replaces it with a new Jacobian matrix **J**' that is calculated using the equation

$$\mathbf{J}' = \mathbf{J} \circ \mathbf{L} \tag{9.9}$$

In equation 9.9, "°" represents the Schur matrix product; this denotes multiplication of corresponding elements of each matrix on the right side of the equation to form the matrix on the left side of the equation. For **L** to be a valid localization matrix, each of its elements must lie in the range [0,1].

If asked to do so, PEST_HP can compute its own localization matrix (and re-compute it during each iteration of the inversion process) using an "adaptive localization" methodology that is similar to that described by Luo and Bhakta (2020). However instead of applying this localization matrix to the Jacobian matrix, PEST_HP applies it to the parameter-to-observation covariance matrix C(**o**,**k**) computed using equation 9.5. (This has the same dimensions as the Jacobian matrix.) First PEST_HP computes a matrix of correlation coefficients between observations and parameters. The empirical correlation coefficient ρ_{ij} between observation *i* and parameter *j* is computed as

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_i^2 \sigma_j^2}} \tag{9.10}$$

where

σ_{ij}	is the empirical covariance between observation *i* and parameter *j* (i.e. element (*i,j*) of C(**o**,**k**));
σ^2_j	is the empirical variance of parameter *j* (i.e. the j'th diagonal element of D(**k**) of equation 9.6); and
σ^2_j	is the empirical variance of observation *i*.

Where there is no correlation between observation *i* and parameter *j*, then the empirical value of ρ_{ij} calculated as above may nevertheless possess a non-zero value. In fact, its expected absolute value is $N^{1/2}$ where *N* is the number of random parameter sets used in construction of the empirical covariance matrices C(**o**,**k**) and D(**k**). If asked to do so, PEST_HP compares the absolute value of each correlation coefficient ρ_{ij} with t_α computed as

$$t_\alpha = \frac{|\rho_{ij}|}{N^{½}} \times 2\ln{(N)} \times A \tag{9.11}$$

Where *A* is the user-supplied value of AUTOTHRESH (see below). If $|\rho_{ij}|$ is less than t_α, the value of σ_{ij} (i.e. C(**o**,**k**)$_{ij}$) is reduced, as this covariance may actually be spurious rather than real as an outcome of the fact that *N* is finite. If "hard autolocalization" is requested by a user, then σ_{ij} is set to zero. If "soft localization" is requested by the user, then σ_{ij} is multiplied by a value between zero and one using the Gaspari-Cohn tapering function; see Luo and Bhakta (2020) for more details (including a description of this function). The value of this tapering function is 1.0 when $|\rho_{ij}|$ is greater than or equal to t_α. It diminishes smoothly to zero as $|\rho_{ij}|$ falls to zero.

A value of between 0.1 and 1.0 is often appropriate for AUTOTHRESH. The use of this autolocalization threshold value is based on the premise that empirical correlation coefficients of less than t_α may be indicative of random noise, and hence may not be real. The corresponding element of C(**o**,**k**) is therefore replaced with a value of zero, or with a value of diminished magnitude. Hence only "real", and not spurious, parameter-to-observation correlations remain in C(**o**,**k**) and are therefore used to calculate **J**.

At the end of each iteration of an inversion process in which autolocalization is implemented, PEST_HP appends information to an "autolocalization report file". This file is named *case.alr*, where *case* is the filename base of the PEST control file. The autolocalization report file contains three columns of information. The first column specifies the iteration number. The second column specifies the number of elements contained in the C(**o**,**k**) matrix; this does not change from iteration to iteration. The third column specifies the number of elements within this matrix for which corresponding parameter-to-observation correlation coefficients are less than t_α. Corresponding elements of C(**o**,**k**) are either set to zero (for hard autolocalization) or diminished in value (for soft autolocalization). An inspection of the autolocalization report file may assist a user in choosing an appropriate value for t_α. The number of elements of C(**o**,**k**) whose values are diminished is also reported to the screen.

9.4 Randomized Jacobian Matrix: Practice

9.4.1 General

As has already been discussed, PEST_HP can generate random parameter increments itself. Alternatively, it can accept random parameter increments generated by another program. Each of these alternatives is discussed below.

Regardless of how they are generated, a modeller is faced with the choice of how many random parameter sets *N* to use in building a diminished rank approximation to the Jacobian matrix. There are two factors at play here. The larger is *N*, the better will the empirical covariance matrix computed using equation 9.4 approximate the true covariance matrix. This is because spurious parameter-to-observation correlations are reduced as *N* increases; as has already been discussed, they are reduced in proportion to $N^{½}$. On the other hand, the whole purpose behind construction of a randomized Jacobian matrix is to reduce the numerical burden of parameter estimation. Therefore, *N* should ideally be significantly smaller than the number of adjustable parameters. In practice, *N* must be at least as large as the dimensionality of the calibration solution space; theoretically, it should be slightly larger than this, as a small amount of parameter redundancy provides a better guarantee that the dimensionality of the range space of **J** is properly defined by model outcomes calculated using these increments.

It is up to the modeller to choose a value for *N* that reconciles these two conflicting demands. The problem of choosing *N* is exacerbated by the fact that the dimensionality of the calibration solution space (and hence the Jacobian matrix range space) may not be known in advance. PEST_HP accommodates uncertainty in choosing a suitable value for *N* by allowing it to increase, on an as-needed basis, as the inversion process progresses.

9.4.2 PEST_HP-Generated Parameter Increments

PEST_HP employs a normal probability distribution to generate random realizations of parameters on which to base model runs to fill the Jacobian matrix. The mean value of each parameter is its current reference value (i.e. the best parameter value inherited from the parameter upgrade process conducted during the previous iteration). Hence the mean value of parameter increments is zero. Parameter increment standard deviations are based on variables that would otherwise be used to control finite-difference derivatives calculation.

Ideally, standard deviations employed in generation of random parameter increment values should be small. However if they are too small, then round-off errors can diminish the integrity of variances and covariances calculated using equations 9.5 and 9.6. Numerical errors in C(**o**,**k**) can be exacerbated by model solver convergence difficulties; the resulting “model output noise” can impair the numerical integrity of small differences of large quantities. In generating parameter realizations, PEST_HP establishes the standard deviation for each parameter as the increment that the parameter would have if two-point (rather than three or five point) finite-difference derivatives were employed to fill the Jacobian matrix. The PEST control variables which determine these increments pertain to parameter groups, and are thus recorded in the “parameter groups” section of a PEST control file. These are the DERINC, DERINCLB, and INCTYP variables. However PEST_HP allows parameter standard deviations calculated in this way to be multiplied by a user-specified factor that is applied to all parameter groups. Hence they can be made as large or as small as a user desires. Alternatively, values for DERINC, DERINCLB and INCTYP can be chosen specifically to achieve desired standard deviations.

When calculating randomly-incremented values for parameters, PEST_HP ensures that these values do not transgress parameter bounds. If, at a particular stage of the inversion process, the current reference value for a parameter is at its upper or lower bound, the sign of the random parameter increment is reversed if necessary to ensure that the parameter's increment moves its value away from this bound rather than across it. On the other hand, if a parameter is not currently at its bound, but its incremented value lies above the parameter's upper bound or below its lower bound, then the value ascribed to the randomly-incremented parameter is adjusted to coincide with the bound.

9.4.3 User-Supplied Parameter Increments

PEST_HP can be directed to one or a number of so-called "JCB files" to read parameter increment realizations. These are binary files that are produced by PEST-suite programs such as RANDPAR3 and RANDPAR4. Their contents can be examined and manipulated by other PEST-suite programs such as JCB2CSV, JCB2PAR and JCB2RRF.

Certain requirements must be met by a random parameter increment JCB file that is supplied to PEST_HP. PEST_HP can check for some of these, but not for all of them.

First, every parameter that is declared as adjustable in the PEST control file on which the PEST_HP inversion process is based must be featured in the user-supplied JCB file. However, there is no need to supply realizations of fixed or tied parameters in these JCB files. If values for these parameters are in fact supplied, they are ignored by PEST_HP, for fixed parameters are not incremented, and tied parameters are calculated from the values of their parent parameters.

Second, as stated above, because the realizations that are supplied in a JCB file are considered to be increments of current parameter values, they should be centred on one or zero – one for log-transformed parameters and zero for non-log-transformed parameters. For log-transformed parameters, parameter increments are actually multipliers. Hence if increments are generated using the RANDPAR3 utility, the PEST control file on which the running of RANDPAR3 is based should provide initial parameter values that are zero or one as appropriate. Because RANDPAR4 takes care of this automatically, its use is preferable to that of RANDPAR3 when generating random realizations of parameter increments.

It is often convenient to base generation of random parameter increments on the prior parameter probability distribution. However it must be born in mind that prior parameter standard deviations and variances may be too large to use as parameter increments for the purpose of Jacobian matrix computation. PEST_HP allows a uniform factor to be applied to all increments that it reads from a JCB file in order to reduce their values, if a user considers this necessary.

If the value of any adjustable parameter, after adjustment by an externally-supplied increment, exceeds its upper bound, then the value of that parameter is clipped at its bound. However tied parameters are not clipped at their bounds; they continue to respect the same ratio with their parent parameters as that provided in the PEST control file on which the PEST_HP run is based.

9.4.4 Jacobian Matrix Retainment

Optionally, the Jacobian matrix calculated during a given iteration of the PEST_HP inversion process can incorporate part of the Jacobian matrix calculated during the previous iteration of that process. This may reduce errors in this matrix incurred by use of a small number of random parameter increment realizations in its construction. This strategy can also reduce calibration-induced parameter and predictive bias, as it allows the rank of the Jacobian matrix to grow over time as its current contents are based on an ever-increasing number of random parameter increment realizations. However there may be a price to be paid for mixing the current Jacobian matrix with previously-calculated Jacobian matrices. For a highly nonlinear model, and/or for an inverse problem that is profoundly ill-posed, this strategy may compromise PEST_HP's ability to calculate up-to-date Jacobian matrices which support continuous objective function improvement in difficult objective function terrains.

PEST_HP reads a user-supplied control variable named RANDJACRETAIN. This specifies the fraction of the previous Jacobian matrix that is retained in calculating a new Jacobian matrix. Suppose that a rank-deficient Jacobian matrix has just been calculated using equation 9.4 (possibly modified by localization functionality as described above). Let this matrix be denoted as $\mathbf{J}_1$. Let $\mathbf{J}_2$ specify the Jacobian matrix (unimproved by Broyden updating) that was calculated during the previous iteration (which, of course, includes part of the Jacobian matrix calculated during the iteration before that). Then the pre-Broyden-updated Jacobian matrix **J** used in the current iteration is calculated as:

$$\mathbf{J} = \frac{\mathbf{J}_1 + R \times \mathbf{J}_2}{1+R} \tag{9.12}$$

Where *R* is the value of RANDJACRETAIN. As in normal usage of PEST_HP, a modeller can implement Broyden updating of a Jacobian matrix based on model outputs calculated during a single, parallelized round of parameter upgrade testing. Theoretically, deficiencies in this matrix can thereby be partially rectified. A second round of parallelized parameter upgrade testing can then be undertaken using the improved Jacobian matrix. In some circumstances this may result in the calculation of parameter upgrade vectors that effect a larger improvement in the objective function than those calculated using a non-Broyden-updated Jacobian matrix. In other cases it does not. Nevertheless, Broyden updating of the Jacobian matrix is recommended, for although its use incurs a model run cost, experience suggests that it can often get a stalled PEST_HP run moving again.

9.5 JCO and JCR Files

As is usual practice, PEST_HP records a JCO file (i.e. a Jacobian matrix file) at the end of each iteration of the inversion process in which parameters are improved. Hence the JCO file that is present at the end of the inversion process (and at any stage thereof) is that which was used to calculate parameters appearing in the PAR file (i.e. the parameter value file). The filename base of both of these files is that of the PEST control file. The PAR file records best parameters achieved up to any time in the inversion process.

When PEST_HP employs its randomized Jacobian functionality, the composition of the Jacobian matrix that is stored in the JCO file may reflect all of the following processes:

1. The outcomes of equation 9.4 based on the set of random parameter increments used in the most recent iteration of the inversion process;

2. The outcomes of inclusion of a fraction RANDJACRETAIN of the non-Broyden-upgraded Jacobian matrix used in the previous iteration of the inversion process;
3. If undertaken, Broyden improvement of the Jacobian matrix based on a parallelized set of model runs in which upgraded parameters are tested.

The format of a JCR file is identical to that of a JCO file. So, after being renamed with a "*.jco*" extension, it can be read, and its contents manipulated, by PEST-suite software which is designed for this purpose. However there are two major differences between a JCR file written by PEST_HP and a JCO file. They are as follows:

1. The Jacobian matrix recorded in a JCR file is not Broyden-updated;
2. The JCR file is updated during every iteration of the inversion process, regardless of whether or not parameters were improved during that iteration. It comprises the **J** of equation 9.12; in doing so, it includes the effects of automatic and/or user-specified localization if these are employed in the PEST_HP inversion process.

As is reported in PEST documentation, Jacobian matrices have many uses. One of these uses is linear parameter and predictive uncertainty analysis. Whether a rank-deficient Jacobian matrix calculated using the randomized Jacobian functionality of PEST_HP is suitable for use in linear analysis is a matter that is currently under investigation. However if it can be so used, the Jacobian matrix contained in a JCR file may be more suitable for this purpose than that contained in a JCO file. This is because the Broyden upgrading procedure may be very effective in promulgating parameter improvements. However it may distort the Jacobian matrix to some extent.

9.6 Control Variables

9.6.1 Reading the Control Variables

Variables which control the action of PEST_HP's randomized Jacobian functionality are read from a section of the PEST control file that is dedicated to holding these variables. Unsurprisingly, this section is named the "randomized jacobian" section. This section must have at least five lines, the first being a header line. The variables comprising these lines are listed in figure 9.1. An example is provided in figure 9.2.

```
* randomized jacobian
RANDOMJAC  RANDOMSEED
NRANDSETSTART NRANDSETINC NRANDSETFIN PHIREDRANDINC RANDINCFAC
RANDJACRETAIN [AUTOLOC AUTOTHRESH LASTLOCITN LOCJCOFILE]
NUMINCSCHED   RANDSCHEDITN1    RANDSCHEDITN2
The following line must be repeated NUMINCSCHED times.
NUMITN  FRACORIGPHI  INCJCBFILE  RANDINCFAC  RANDJACRETAIN
```

Figure 9.1 Variables appearing in the "randomized jacobian" section of a PEST control file.

```
* randomized jacobian
1    222                  #RANDOMJAC  RANDOMSEED
80  5  100  0.1  2.0      #NRANDSETSTART NRANDSETINC NRANDSETFIN PHIREDRANDINC RANDINCFAC
0.4 2 1.0 5               #RANDJACRETAIN AUTOLOC AUTOTHRESH LASTLOCITN (no LOCJCOFILE)
4     1       999         #NUMINCSCHED   RANDSCHEDITN1    RANDSCHEDITN2
10   0.5    inc1.jcb  0.1  0.05  #NUMITN  FRACORIGPHI  INCJCBFILE  RANDINCFAC RANDJACRETAIN
10   0.1    inc2.jcb  0.1  0.1   #NUMITN  FRACORIGPHI  INCJCBFILE  RANDINCFAC RANDJACRETAIN
100  0.005  fdinc     2.0  0.2   #NUMITN  FRACORIGPHI  INCJCBFILE  RANDINCFAC RANDJACRETAIN
100  0.001  fdinc     2.0  0.3   #NUMITN  FRACORIGPHI  INCJCBFILE  RANDINCFAC RANDJACRETAIN
```

Figure 9.2 An example of the "randomized jacobian" section of a PEST control file.

The “randomized jacobian” section must be placed between the “control data” and “parameter groups” sections of a PEST control file. Other sections (such as “singular value decomposition”, “simultaneous parameter increments”, “sensitivity reuse” and “rsi”) can also appear between the “control data” and “parameter groups” sections of a PEST control file. There is no prescribed ordering for these sections if more than one of them is present.

Note that, at the time of writing, PESTCHEK tolerates the presence of a “randomized Jacobian” section in a PEST control file, but does not check its contents. The PSTCLEAN utility removes this section from a PEST control file in order to render that file inoffensive to PEST and BEOPEST.

The roles of variables appearing in the “randomized Jacobian” section of a PEST control file are now discussed in detail.

9.6.2 RANDOMJAC

This variable (an integer) must be set to either 0 or 1. If it is set to 0, then randomized Jacobian calculation is disabled; hence derivatives are calculated using finite parameter differences in the normal PEST_HP manner. Alternatively, if RANDOMJAC is set to 1, then randomized Jacobian calculation is enabled.

9.6.3 RANDOMSEED

RANDOMSEED is the seed for the random number generator. Select any integer greater than zero.

9.6.4 NRANDSETSTART, NRANDSETINC, NRANDSETFIN, PHIREDRANDINC

The first three of these variables are integers while the last is a real number. They all pertain to the variable *N* (i.e. the number of parameter realizations) featured in equations 9.5, 9.6 and 9.11.

NRANDSETSTART is the initial value of *N*.

If, during a particular iteration of the inversion process, the value of the (measurement) objective function does not improve by more than a relative amount of PHIREDRANDINC of its best value achieved to date, then PEST_HP increases *N* by NRANDSETINC (“INC” stands for “increment”). However *N* will never be allowed to increase above NRANDSETFIN (“FIN” stands for “finish”). A value of 0.1 is often suitable for PHIREDRANDINC.

Some of the matters which need to be considered when selecting values for NRANDSETSTART, NRANDSETINC and NRANDSETFIN are discussed earlier in this chapter. In addition to the theoretical and numerical considerations that were discussed above, a further practical matter requires consideration. PEST_HP commissions model runs on a suite of agent nodes. Sometimes these nodes will have been purchased from a cloud provider. In these inversion circumstances, under-utilization of agent nodes serves no numerical purpose, but increases financial costs. Hence, at any stage of the inversion process, the value of *N* should be an integral multiple of the number of available agents.

9.6.5 RANDINCFAC

If PEST_HP generates realizations of random parameter increments itself, it declares all adjustable parameters to be statistically independent. The mean value of each parameter is its reference value at that particular stage of the inversion process. The standard deviation ascribed to each parameter is equal to the increment that it would have been assigned if derivatives were computed using two-point finite parameter differences. As stated above, these increments are controlled by the DERINC, DERINCLB and INCTYPE control variables that are assigned to parameter groups.

Standard deviations calculated for all parameters (and hence all parameter increments) using the DERINC, DERINCLB and INCTYPE control variables are multiplied by RANDINCFAC prior to the random number generation process. RANDINCFAC is a real number whose value must be greater than zero.

9.6.6 RANDJACRETAIN

This real variable should be set to a value between 0.0 and 1.0. It governs how much of the covariance matrix calculated during the previous iteration of the inversion process is retained for use in calculating parameter upgrades during the current iteration; see equation 9.12. Set RANDJACRETAIN to 0.0 to suppress carryover of the Jacobian matrix between iterations.

Note that, as was stated above, the Jacobian matrix from a previous iteration that is partially retained for use in the current iteration has not undergone Broyden upgrading, even if Broyden upgrading of that matrix was implemented during the previous iteration.

9.6.7 AUTOLOC, AUTOTHRESH and LASTLOCITN

AUTOLOC is an integer variable. If it is set to zero, then autolocalization is disabled. A setting of 1 enables hard autolocalization, while a setting of 2 enables soft autolocalization.

AUTOTHRESH is the value of A of equation 9.11 The recommended value of this variable is between 0.1 and 1.0. The lower that AUTOTHRESH is set, the less aggressive is autolocalization; conversely, setting it to a higher value is more aggressive as it results in reduction in the absolute values of a greater number of Jacobian elements (because a greater number of observation-to-parameter correlations fall below the t_α threshold of equation 9.11). The behaviour of PEST_HP in lowering the (measurement) objective function, and an inspection of the autolocalization report file (*case.alr*) may suggest that it be raised or lowered from its existing value. Recall that *case.alr* records the number of parameter-to-observation correlation coefficients that are less than the t_α threshold. Corresponding elements of C(**o**,**k**) are set to zero if hard thresholding is enabled, while their values are diminished if soft thresholding is enabled. PEST_HP also records the number of affected Jacobian elements on the screen.

If activated by setting AUTOLOC to 1 or 2, automatic localization is implemented during every iteration of the inversion process using the outcomes of the most recent set of model runs; these are the model runs used for construction of the current C(**o**,**k**) parameter-to-observation covariance matrix. However automatic localization is not implemented for iterations LASTLOCITN+1 and later. Set LASTLOCITN to a large value (e.g. 999) to ensure that autolocalization is implemented throughout the inversion process.

AUTOLOC can be omitted from the "randomized jacobian" section of a PEST control file, so that RANDJACRETAIN is the only variable featured on the fourth line of this section. In this case, AUTOLOC is assumed to be 0. If it is explicitly or implicitly set to zero, then entries for AUTOTHRESH and LASTLOCITN can be omitted. However values for all of these variables must be supplied if a filename is supplied for the LOCJCOFILE variable, for this is (optionally) identified as the fifth item on this line.

9.6.8 LOCJCOFILE

If a user wishes that PEST_HP employ a localization matrix that he/she has prepared him/herself (i.e. the matrix **L** of equation 9.9), then the name of a file containing this matrix must be supplied as the value of the LOCJCOFILE variable. This must be a JCO file. It may have been written by a program such as JCOBLANK. The matrix contained in this file must have the following specifications:

- It must have dimensions of NOBS by NESPAR, where NOBS is the number of observations (excluding prior information) featured in the current PEST control file, and NESPAR is the number of adjustable parameters featured in the current PEST control file.
- All elements of this matrix must be between zero and one (inclusive).

PEST_HP will object with an error message if either of these conditions is violated.

The localization matrix provided in the LOCJCOFILE file is used for all iterations of the inversion process, regardless of the setting of LASTLOCITN. It can then be used instead of, or as well as, autolocalization.

9.6.9 NUMINCSCHED

NUMINCSCHED stands for "number of increment schedules". An "increment schedule" instructs PEST_HP how to obtain random parameter increments during a particular iteration of the inversion process. As has already been discussed, PEST_HP can calculate increments itself; alternatively it can read them from a user-supplied JCB file. A user can supply a number of different JCB files containing increment realizations with different statistical characteristics. Each of these must be assigned to a different increment schedule.

If NUMINCSCHED is set to zero, then no parameter increment schedules are provided. Hence, throughout the entire inversion process, PEST_HP calculates parameter increments itself based on the assumption of parameter statistical independence using variables supplied in the "parameter groups" section of the PEST control file to express parameter increment stochasticity.

9.6.10 RANDSCHEDITN1 and RANDSCHEDITN2

These are both integer variables They must both be zero or greater. If one of them is zero, then the other must be zero; otherwise RANDSCHEDITN2 must be greater than RANDSCHEDITN1. If RANDSCHEDITN1 and RANDSCHEDITN2 are both greater than zero, then NUMINCSCHED must be greater than 1.

Suppose that RANDSCHEDITN1 is set to i_1 and that RANDSCHEDITN2 is set to i_2. Then during iterations i_1 to i_2 of the inversion process, PEST_HP selects an increment schedule randomly from the first NUMINCSCHED-1 schedules that are provided to it. On iterations following i_2, PEST_HP employs the last parameter increment schedule (i.e. schedule NUMINCSCHED) exclusively. Set RANDSCHEDITN2 to a high number such as 999 if you do not wish that PEST_HP makes exclusive use of this final increment schedule as it approaches the end of the inversion process.

If RANDSCHEDITN1 and RANDSCHEDITN2 are both set to zero, then PEST_HP uses increment schedules in order of their listing in the "randomized Jacobian" section of the PEST control file, changing from one to the next in accordance with the settings of pertinent increment schedule control variables.

9.6.11 INCJCBFILE

INCJCBFILE is a character variable. Optionally, it contains the name of a JCB file which houses an ensemble of parameter increment realizations. (Note that a JCB file must possess an extension of

"*.jcb*".) As has already been stated, this file will normally have been written by the RANDPAR4 utility. Increments should be centred on 0.0 for non-log-transformed parameters and on 1.0 for log-transformed parameters. There should be at least as many increment realizations in this file as PEST_HP needs to use. Hence if, for example, this file will be used over four iterations of the inversion process and the NRANDSETFIN control variable is set to 200, then this JCB file should contain at least 800 realizations of parameter increments. If RANDSCHEDITN1 and RANDSCHEDITN2 are greater than zero, then the number of iterations for which PEST_HP requires increments from this file must be guessed, as use of this file is a random matter. It is better to over-supply increment realizations than to under-supply them; this is not a problem if using RANDPAR4 to fill this file.

Alternatively, INCJCBFILE can be supplied as "fdinc". This instructs PEST_HP to calculate parameter increments itself under the assumption of parameter stochastic independence using variables which would otherwise control the calculation of finite difference derivatives.

9.6.12 RANDINCFAC and RANDJACRETAIN

All increments read from file INCJCBFILE, or their logs, are multiplied by an increment-schedule-specific value of RANDINCFAC before being used by PEST_HP. (Their increments are multiplied if a parameter is untransformed, and their logs are multiplied if a parameter is log-transformed.) Alternatively, if INCJCBFILE is set to "fdinc" so that PEST_HP calculates increments itself, then RANDINCFAC multiplies these increments. This overrides the value supplied for RANDINCFAC on the third line of the "randomized jacobian" section of the PEST control file. The latter variable is employed only if NUMINCSCHED is set to zero.

An increment-schedule-specific fraction RANDJACRETAIN of the Jacobian matrix used in the previous iteration is retained in calculating the Jacobian matrix on which parameter upgrades are calculated during the current iteration. This overrides the value supplied for RANDJACRETAIN on the fourth line of the "randomized jacobian" section of the PEST control file. The latter variable is employed only if NUMINCSCHED is set to zero.

9.6.13 NUMITN and FRACORIGPHI

If RANDSCHEDITN1 and RANDSCHEDITN2 are both set to zero, then PEST_HP uses increment schedules in the order in which they are provided in the "randomized jacobian" section of the PEST control file. It uses a particular increment schedule for NUMITN iterations, or until the (measurement) objective function has fallen to a fraction FRACORIGPHI of its original value. Then it progresses to use of the next increment schedule. Note that, in order for this type of increment scheduling to work, values provided for FRACORIGPHI must reduce with increasing increment schedule number.

9.7 Accommodating Other PEST_HP Functionality

9.7.1 Incompatibilities

Randomized Jacobian functionality cannot be employed under any of the following circumstances:

- PEST_HP is undertaking SVD-assisted parameter estimation, or is run in "pareto" mode;
- The PEST_HP control file instructs PEST_HP to employ different commands for running the model when calculating finite-difference derivatives with respect to different parameters;
- Split slope analysis is used for accommodation of poor numerical derivatives;
- Sensitivity re-use functionality has been enabled;

- Parameter bounds sticking functionality has been enabled.

If you attempt to violate the first two of the above conditions, PEST_HP will cease execution with an error message. In the other three cases, PEST_HP disables the offending functionality itself.

The PEST whisperer (PWHISP_HP) is not yet programmed to read or understand the contents of a PEST run record file recorded by PEST_HP when employing randomized filling of the Jacobian matrix. Nor can the PCOST_HP cost calculator estimate the number of model runs required for parameter estimation based on a randomized Jacobian matrix.

9.7.2 DERFORGIVE

If, during the Jacobian-filling phase of an inversion iteration, the model fails to run to completion while employing a random set of parameter increments, PEST_HP does not cease execution with a pertinent error message. In accommodating model run failure in this way, its behaviour is thus similar to finite-difference filling of the Jacobian matrix with a DERFORIGVE setting of “derforgive”. Because the model produces no useable outputs under these circumstances, the covariance matrix of equation 9.5 is not updated in the way that it would be updated if the model run was a success. Because a realization is missing, the accuracy of this matrix is lowered; at the same time, its rank-deficiency is increased, as is the subspace of parameter space in which parameter upgrade vectors may lie.

If, on any iteration of the inversion process, all Jacobian-filling model runs fail, PEST_HP ceases execution with an appropriate error message.

9.8 Experience to Date

9.8.1 Autolocalization

History-matching based on randomized Jacobian calculation appears to benefit from the use of hard or soft autolocalization, with AUTOTHRESH set to between 0.1 and 1.0. This is especially the case if using independent random parameter increments, that is if NUMINCSHED is set to zero. With autolocalization switched on, progress during early stages of an inversion process is often greatly accelerated. However it does not appear to be so beneficial towards the end of the process. Hence a setting of about 5 for LASTLOCITN is often beneficial.

9.8.2 Increment Schedules

Use of a randomly-selected mixture of increment schedules appears to work well. Some of these may be based on alternative concepts of the prior parameter probability distribution, and inter-parameter correlations implied by these. Another increment schedule may rely on independent parameter stochasticity (in which case INCJCBFILE for that schedule may be set to “fdinc”); this may support introduction of locally connected permeability (or locally connected impermeability) to the model domain in ways that are required to fit the calibration dataset, but that were unanticipated by the modeller. If RANDSCHEDITN2 is not set to a large number such as 999, then it may be a good idea to set INCJCBFILE to “fdinit” for the final increment schedule; this allows PEST_HP to spend the last few iterations of the inversion process “filling in the fine detail” of a parameter field in order that model outputs can fit the calibration dataset well.

9.8.3 Solution Method

PEST_HP offers the following methods for calculation of upgraded parameters (see PEST documentation for details):

- Gaussian elimination;
- Singular value decomposition with SVDMODE set to 1;
- Singular value decomposition with SVDMODE set to 2;
- LSQR.

As is discussed elsewhere, use of SVD or LSQR ensures stability of the inversion process, regardless of whether or not the inverse problem is well-posed. As has already been discussed, evidence to date (and some theory), suggests that if NUMINCSHED is set to zero so that increment calculation is based solely on use of finite difference control variables (thus assuming stochastic independence of parameter increments), an SVDMODE setting of 2 works well. However this does not appear to be the case where increment ensembles are based on covariance matrices that are more reflective of possible prior parameter distributions.

9.8.4 Tikhonov Regularisation

Tikhonov regularisation allows expert knowledge to enhance the integrity of the inversion process. Its use can promulgate solution of an inverse problem which approaches that of minimum error variance. This often comprises a “smooth” rather than “bumpy” estimated parameter field. A NUMINCSCHED setting of zero can precipitate the emergence of bumpy parameter fields; however the use of Tikhonov regularization, in conjunction with a covariance matrix applied to pertinent prior information equations, can forestall this occurrence.

As is extensively discussed in PEST documentation, in its implementation of Tikhonov regularisation PEST calculates a factor for regularisation weights which attempts to ensure that the measurement objective function attains, but does not undercut, a user-specified target measurement objective function. This weight factor is re-computed during every iteration of the inversion process. Local linearization of the model (i.e. a Jacobian matrix) is required for calculation of this weight factor. Where the Jacobian matrix is only approximate, the weight factor can only be approximate. Furthermore, its value may undergo significant apparent changes from iteration to iteration, this reflecting random variation of elements of the Jacobian matrix.

Experience to date suggests that PEST_HP performs satisfactorily when using a randomized Jacobian matrix if an inverse problem includes Tikhonov regularisation. However while PEST_HP may be able to achieve an appropriate target measurement objective function, it may not be able to lower the regularisation objective function to as low a value as it could if the Jacobian matrix were filled using finite parameter differences.

Experience has also shown that this problem can be ameliorated to some extent if the PHIMACCEPT regularisation control variable is set a little higher than when using finite-difference derivatives for filling of the Jacobian matrix – perhaps 5 percent to 10 percent higher than PHIMLIM. Once the measurement objective function has fallen below PHIMACCEPT, PEST_HP then concentrates on lowering the regularisation objective function. This can result in a much more acceptable parameter field than would otherwise be the case. In particular, the estimated parameter field may not be as "bumpy" as it would otherwise be because of null space entrainment - a problem to which use of a randomized Jacobian is particularly susceptible. (See the discussion at the beginning of this chapter.)

An alternative to determination of a regularisation weight factor based on the value of PHIMLIM is to use the REG2MEASRAT regularisation control variable as a basis for calculation of this factor. However, while this is fast, it is also approximate. Nevertheless a simplified calculation of the regularisation weight factor in this manner is less prone to errors incurred by use of a rank-deficient Jacobian matrix than a more complex calculation based on PHIMLIM. In fact, the Jacobian matrix is not used for calculating the regularisation weight factor at all when the latter's calculation is based on REG2MEASRAT, except perhaps in the first iteration of the inversion process; see documentation of this variable elsewhere in this manual.

Yet another alternative which appears to work well in some circumstances is to set the LAMBDA_FOR_REG control variable to 1 in the "regularisation" section of the PEST control file. Under these circumstances, PEST_HP's selection of the regularisation weight factor is experimental rather than prescriptive. See the pertinent section of this manual for further details.

9.8.5 Retainment of the Previous Jacobian Matrix

For both practical and conceptual reasons, RANDJACRETAIN should be set to a value that allows the Jacobian matrix used for calculation of parameter upgrades in any particular iteration to inherit some characteristics of previously-calculated Jacobian matrices. If this is not done, the rank-deficient matrix **J** calculated using equation 9.4 will possess a solution space (orthogonal complement to its null space) that is likely to be somewhat misaligned with the true solution space of the inverse problem, for it can only be comprised of the space that is spanned by the most recent set of random parameter increment vectors. Provided that the number of these parameter increment sets is equal to, or exceeds, the dimensionality of the solution space, solution space misalignment will not necessarily impair the ability of the inversion process to achieve a good fit with a calibration dataset. However considerable entrainment of null space parameter components will be associated with estimation of parameters. Estimated parameters may therefore endure considerable calibration-induced bias, and estimated parameter fields may lose their aesthetic appeal. To the extent that model predictions show null space dependency, they too may exhibit calibration-induced bias.

By accumulating Jacobian matrices calculated using different realizations of parameter increments over multiple iterations, the rank-deficiency of the Jacobian matrix shrinks over time. This allows a PEST_HP inversion process that is based on Jacobian randomization to define a solution space which is more closely aligned with the true solution space of the inverse problem than would be the case if RANDJACRETAIN were set to 0.0.

Based on limited experience to date, RANDJACRETAIN values of between 0.3 and 0.8 appear to work well. A large value for this variable dilutes the ability of the most recent parameter increments to reflect changes in the Jacobian matrix arising from model nonlinearity. A small value may promulgate null-space entrainment for reasons just outlined.

Because retainment of a previous Jacobian matrix has the potential to increase the rank of the Jacobian matrix, the need for autolocalization (if employed) may diminish with iteration count. In fact, if localization is used to set possibly aberrant Jacobian elements to values at or near zero, the noise that is associated with these elements is never given the chance to cancel through mixing with elements of previously-calculated Jacobian matrices. This strengthens the case for setting LASTLOCITN to a value of 4 or 5, so that the effects of autolocalization can diminish as the benefits of Jacobian retainment grow.

9.8.6 Broyden Jacobian Updating

Experience to date suggests that in some inversion contexts Broyden Jacobian updating can enhance PEST_HP's performance dramatically when it employs a randomized Jacobian matrix. However there are other cases where it does not.

If Broyden Jacobian updating is not employed, this can sometimes result in slower, but steadier, progress of the inversion process, particularly when employing Tikhonov regularisation. In the latter case, regularisation objective functions appear to be lower for a given measurement objective function than if Broyden updating of the Jacobian matrix is employed. This, of course, is desirable, as Tikhonov regularisation poses the inverse problem as a constrained minimization problem, in which the regularisation objective function is minimized subject to the measurement objective function attaining its user-specified target value.

Nevertheless, as stated above, the benefits of Broyden Jacobian updating in achieving a low measurement objective function often outweigh its costs. These costs are the increased number of model runs that are required for solution of the inverse problem, and a possibly higher regularisation objective function. In areas of high expected parameter spatial heterogeneity, the ability to reflect the presence of this heterogeneity through achieving a low measurement objective function may far outweigh the benefits of parameter field smoothness.

9.8.7 Parameter Change Limits

Limited experience to date suggests that an inversion process based on a randomized Jacobian matrix may benefit from the imposition of tighter parameter change limits than would otherwise be the case. This prevents the occurrence of large, but erratic, changes in parameter values, particularly during early iterations of an inversion process. Consider employing RELPARMAX and FACPARMAX settings of between 3.0 and 5.0. Unfortunately, this measure may increase the number of model runs required for solution of an inverse problem. Fortunately, however, limited experience to date also suggests that the need for this measure is diminished if autolocalization is employed.

10. Jacobian Blanking and Simultaneous Increments

10.1 General

This section continues the theme of the previous section. It describes PEST_HP functionality that supports building of a Jacobian matrix using fewer model runs than there are adjustable parameters. In the present case, this is achieved by endowing more than one parameter with an incremental variation when undertaking Jacobian-filling model runs. The previous chapter described the use of random parameter increments to achieve model run efficiency. The present chapter focuses on the use of simultaneous parameter increments used for the purpose of finite difference derivatives calculation. It also discusses the forced zeroing of selected elements of a Jacobian matrix.

Simultaneous incrementation (as distinct from random incrementation) of more than a single parameter when undertaking a model run for the purpose of finite-difference derivatives calculation requires “blockiness” of the Jacobian matrix. That is, it requires that the set of model-generated counterparts to observations (these being simply referred to as “observations” herein) that are sensitive to some parameters are different from the set of observations that are sensitive to other parameters. If observation-to-parameter sensitivities are such that they are not completely block-isolated in this way, they can be rendered such through strategic zeroing of insensitive elements of a Jacobian matrix. Regardless of whether it occurs naturally, or through strategic zeroing of insensitive elements, use of simultaneous parameter increments requires concomitant use of a “blanking matrix” to ensure that variation of a particular model output in response to variation of multiple parameters is not misinterpreted as sensitivity of that output to more than a single one of the varied parameters. A blanking matrix can be supplied by the user, or can be calculated internally by PEST_HP.

Strategic blanking of Jacobian matrix elements can also raise the integrity of a Jacobian matrix computed using random parameter increments. As is described in the previous section of this manual, PEST_HP supports the use of random parameter increments through its “randomized Jacobian” functionality. Through the use of random parameter increments, the number of model runs required per iteration of an inversion process can be reduced to slightly greater than that of the dimensionality of the solution space of the inverse problem on which PEST_HP usage is based. However the dramatic decrease in model run requirements accrued through use of a randomized Jacobian matrix is accompanied by certain costs, particularly if parameter increments are calculated internally by PEST_HP under the assumption of stochastic independence. These costs are as follows.

- While progress in lowering an objective function may be high during early stages of an inversion process that is based on randomized Jacobian matrices, progress at later stages of this process may be compromised by the introduction of spurious correlations between some model outputs and some parameters, especially where true sensitivities of some model outputs to some parameters approach zero. The greater the number of random parameter increment vectors that are used to compute the Jacobian matrix, the smaller will be this effect. However use of a large number of random parameter increment vectors incurs a numerical cost. If the number of random parameter increments approaches the number of adjustable parameters, the benefits of using random parameter increments to fill a Jacobian matrix vanish.

- Parameter upgrades are calculated as linear combinations of parameter increment vectors. Where the number of random parameter increment vectors is equal to, or slightly greater than, the dimensionality of the inverse problem solution space, it is highly probable that they collectively span a space onto which any solution space vector has a non-zero projection; hence the objective function can be lowered. However it is most unlikely that an objective-function-reducing parameter upgrade vector, calculated as a linear combination of these random parameter increment vectors, does not have a substantial null space component. The upgrade vector is therefore likely to exhibit calibration-induced bias.

Strategic Jacobian matrix blanking may reduce both of these costs. In doing so, it performs a similar role to that of "localization" in improving the performance of an iterative ensemble smoother.

10.2 The JCOBLANK Utility

10.2.1 Simultaneous Parameter Increments

A utility program named JCOBLANK has been added to the standard PEST suite. As is described in Part II of the PEST manual, this program performs two tasks. They are

1. Construction of a blanking matrix;
2. Design of a model run schedule for finite-difference filling of a Jacobian matrix using simultaneous parameter increments.

The blanking matrix file and simultaneous increment file that are written by JCOBLANK can be read by PEST_HP. Hence PEST_HP can implement a simultaneous parameter increment strategy devised by JCOBLANK.

JCOBLANK can build a blanking matrix in two ways. One option is to employ a "zeroing file". Through this type of file, a user directly identifies Jacobian matrix elements that must be blanked. Alternatively, JCOBLANK can use an existing Jacobian matrix file (i.e. a JCO file) as a basis for blanking; under these circumstances it constructs a blanking matrix that zeroes elements of the Jacobian matrix file whose weighted sensitivities are low compared to other elements of this matrix. In either case, the greater the number of weighted Jacobian elements that are blanked, the greater is the chance that remaining elements of the Jacobian matrix have enough of a "blocky" structure to support design of a model-run-efficient simultaneous parameter increment strategy.

It is important to remember that gains in model run efficiency attained through use of simultaneous parameter increments may compromise the integrity of a Jacobian matrix. This may slow the progress of an inversion process, and may introduce bias to some estimated parameters. This effect is likely to be most significant for highly nonlinear models for which model output sensitivities to parameters are functions of the values of the parameters themselves.

While JCOBLANK is documented in part II of the PEST manual, some aspects of its functionality that are pertinent to PEST_HP's implementation of simultaneous parameter increments are now briefly discussed.

10.2.2 Observation Weights

Ideally, rows of a Jacobian matrix for which observation weights are zero should be blanked when this matrix is used as a basis for the design of a simultaneous parameter increment strategy. This is because rows pertaining to observations with zero weight can be omitted from a Jacobian matrix with no adverse consequences for an inversion process. Blanking of zero-weighted rows prevents non-zero sensitivities in these rows from compromising potential blockiness of the Jacobian matrix, and hence development of an efficient simultaneous parameter increment strategy. JCOBLANK automatically blanks zero-weighted rows of a Jacobian matrix if it builds a blanking matrix from this matrix. However it does not automatically blank zero-weighted Jacobian matrix rows if it follows instructions provided in a zeroing file. It is thus the user's responsibility to ensure that zero-weighted rows are blanked through the instructions which he/she provides in this file if this is his/her desire. There may, indeed, be occasions where blanking of zero-weighted rows is not desirable. This occurs if an inversion process "carries" one or a number of model predictions for which linear uncertainty analysis will later be undertaken.

10.2.3 Simultaneous Increment Strategy

Design of a simultaneous increment strategy can be a numerically intensive procedure. Optimization of this strategy (i.e. reducing the number of model runs required for filling a blocky Jacobian matrix) can be even more numerically intensive. JCOBLANK uses a rather simple algorithm for development of a simultaneous increment strategy. It begins the process by inspecting the first column of the Jacobian matrix. It then determines whether another adjustable parameter can be incremented conjunctively with the parameter associated with this first column by visiting subsequent columns of the Jacobian matrix in order to ascertain whether all non-zero sensitivities in one of these columns are non-overlapping with those of the first column. If it finds such a column, the parameter that is associated with this column is co-assigned to the first Jacobian-building model run. JCOBLANK then continues its inspection of Jacobian matrix columns; however now it looks for a column of the Jacobian matrix whose non-blanked elements have no coincidence with non-blanked elements of columns associated with the two parameters that have been assigned to the first model run. If it finds such a column, then another parameter is designated as being simultaneously incremented during the first Jacobian-building model run. This column-search procedure continues until all Jacobian columns have been visited; the parameter composition of the first model run is thereby completely determined. JCOBLANK then repeats this process to determine the parameter composition of the second model run. And so on.

This strategy is reasonably efficient. However its outcomes may depend on the order in which Jacobian columns are visited. In particular, the parameter composition of different Jacobian filling model runs may be different if Jacobian columns are visited in a random order rather than in a sequential order. In contrast to JCOBLANK, PEST_HP employs a random visitation order of Jacobian matrix columns when it is asked to devise a simultaneous parameter increment strategy (see below). Hence the simultaneous increment strategy which it devises may be different from that devised by JCOBLANK, even if they are based on the same Jacobian matrix. Furthermore, neither of these strategies can be guaranteed to be that which minimizes the total number of simultaneous model runs required for finite-difference filling of a blocky Jacobian matrix.

10.2.4 Blanking Re-Visited

Before it devises a simultaneous parameter increment strategy, JCOBLANK actually builds a blanking matrix based on sensitivities that it finds in the Jacobian matrix. In accordance with user instructions, Jacobian matrix elements with low sensitivities are blanked in order to enhance blockiness of this matrix. The simultaneous increment strategy that it devises is based on this blanking matrix.

Once it has devised a simultaneous parameter increment strategy based on the blanking matrix, JCOBLANK can be asked to review the blanking matrix. The simultaneous parameter increment strategy that it devised using the above algorithm may not require that all blanked elements of a blanking matrix remain blanked. Hence, if asked to do so, JCOBLANK unblanks elements of the blanking matrix whose blanking is nonessential for support of the simultaneous parameter incrementation strategy which it has just devised. PEST_HP does this automatically when it devises its own simultaneous parameter increment strategy; see below.

10.2.5 Multiple Command Lines

If the NUMCOM variable in the "control data" section of a PEST control file is set to a number greater than 1, then PEST and PEST_HP employ different commands to run a model when calculating sensitivities with respect to different parameters. The index of the command that is used to run the model for a specific parameter is supplied through the DERCOM variable that is associated with each parameter in the "parameter data" section of the PEST control file.

If a model employs multiple command lines, the simultaneous parameter increment strategy devised by JCOBLANK and PEST_HP respect this. That is, only parameters that employ the same model command are incremented simultaneously.

10.2.6 Prior Information

Both JCOBLANK and PEST_HP ignore prior information when they build a blanking matrix, and then devise a simultaneous parameter increment strategy on the basis of that matrix. Because sensitivities of prior information equations are directly supplied through the prior information equations themselves, they do not require calculation through finite parameter differencing. Efficiency of their calculation does not therefore benefit from a simultaneous parameter increment strategy.

10.2.7 Covariance Matrices

If a covariance matrix is assigned to an observation group which does not exclusively pertain to prior information, then JCOBLANK does not perform Jacobian blanking and evaluation of a simultaneous parameter increment strategy if it is asked to base these on an existing Jacobian matrix. This is because JCOBLANK actually works with a weighted Jacobian matrix under these circumstances. The relationship between Jacobian rows and weights is invalidated by the use of a covariance matrix that applies to multiple observations.

10.3 PEST_HP and Simultaneous Parameter Increments

PEST_HP can read a blanking matrix from a so-called "blanking file". It can also read a file containing a simultaneous parameter increment strategy from a so-called "simultaneous increment file". These files must complement each other; they may both have been written by JCOBLANK. PEST_HP uses the contents of the simultaneous increment file to allocate parameter increments to Jacobian-filling model runs. Then, after the Jacobian matrix has been filled using these model runs, it blanks this matrix using information obtained from the blanking file; as stated above, this eliminates confusion in attribution of model output sensitivities to parameters.

Alternatively, PEST_HP can itself be asked to create a blanking matrix itself based on the latest Jacobian matrix that it has calculated, and to formulate a simultaneous parameter increment strategy based on this blanking matrix. These are used to assist it in building the next Jacobian matrix. PEST_HP constructs the blanking matrix by identifying elements of low absolute value in the Jacobian matrix that it has just filled; in doing this, it employs an identical algorithm to that used by JCOBLANK when it is asked to build a blanking file based on an existing Jacobian matrix. After it has formulated a simultaneous parameter increment strategy based on this blanking matrix, PEST_HP saves a blanking file and a simultaneous increment file for use in the next iteration of the inversion process (and possibly in iterations following that).

Note the following aspects of PEST_HP's behaviour in building a blanking matrix, and in designing a simultaneous parameter increment strategy.

1. Rows of the Jacobian matrix corresponding to observations that are assigned weights of zero are blanked. If a PEST control file is "carrying" zero-weighted observations as predictions for later use in linear predictive uncertainty analysis, the blanking of these rows of the Jacobian matrix will invalidate this analysis.
2. PEST_HP's algorithm for construction of a simultaneous parameter increment scheme is similar to that of JCOBLANK. However in determining which parameters to increment on a particular model run, it visits columns of the Jacobian matrix in random order. Furthermore, the visitation order varies between the different occasions on which it devises a simultaneous parameter increment strategy.
3. The random number seed which governs PEST_HP's random column visitation order is set internally. It can be re-set by providing a seed to the RANDOMSEED control variable in the "randomized jacobian" section of a PEST control file. This does not require activation of randomized Jacobian functionality; RANDOMJAC can be set to zero in this section of the file, thereby de-activating this functionality.
4. Because PEST_HP visits columns of the Jacobian matrix in random order and JCOBLANK visits these columns in sequential order, a simultaneous parameter increment strategy devised by PEST_HP may differ from that devised by JCOBLANK if the latter program is provided with the same Jacobian matrix.
5. Once it has devised a simultaneous parameter increment strategy, PEST_HP alters the blanking matrix so that it conforms with this strategy (see above). Hence no blanking takes place beyond that which is required to complement the simultaneous parameter increment strategy.
6. If simultaneous parameter increments are employed in building a Jacobian matrix, PEST_HP disables sensitivity re-use and parameter bounds sticking if either of these have been enabled through settings supplied in a PEST control file.
7. As is usual practice, if the NOPTMAX control variable appearing in the "control data" section of a PEST control file is set to -1 or -2, PEST_HP ceases execution after filling the Jacobian matrix. If the SIMINCCALC control variable (see below) is set to 1, then PEST_HP also constructs a blanking matrix and devises a simultaneous parameter increment strategy before ceasing execution; as is described below, these are stored in appropriately-named blanking and simultaneous increment files. If desired, these files can be used on subsequent PEST_HP runs in which the SIMINC control variable is set to 1; however they must first be renamed (see below).

Note that it is not always necessary for Jacobian matrix blanking to be undertaken as part of the design of a simultaneous parameter increment strategy. As stated above, Jacobian matrix blanking may sometimes be undertaken to simply improve the effectiveness of randomized Jacobian matrix calculation; set JCOBLANK to -1, and both SIMINC and SIMINCCALC to 0 to achieve this outcome (see below). With these settings PEST_HP does not record a blanking matrix in an external file; nor does it design a simultaneous increment strategy. It simply blanks any Jacobian matrix that it calculates, and reports to the screen the number of Jacobian elements that it blanks.

10.4 PEST_HP Control Variables

10.4.1 Section in PEST Control File

A new section has been introduced to the PEST_HP control file. This section is labelled "* simultaneous parameter increments". It must be placed between the "control data" and "parameter groups" sections of a PEST control file. Other sections (such as "singular value decomposition", "randomized jacobian" and "sensitivity reuse") can also appear between the "control data" and "parameter groups" sections of a PEST control file. There is no prescribed ordering for these sections if more than one of them is present.

Figure 10.1 shows variables which are featured in the "simultaneous parameter increments" section of a PEST_HP control file. Figure 10.2 shows an example.

Note that, at the time of writing, PESTCHEK tolerates the presence of this section, but does not check its contents. The PSTCLEAN utility removes this section from a PEST control file in order to render that file inoffensive to the normal versions of PEST and BEOPEST.

```
* simultaneous parameter increments
BLANKJCO    SIMINC
BLANKFILE
SIFILE
SIMINCCALC
FRACOBS FRACPAR FULLJCOITN SIMINCACCEL MINSIMINC MATCHAGENTS
```

Figure 10.1. Specifications of the "simultaneous parameter increments" section of a PEST control file.

```
* simultaneous parameter increments
1 1                             #BLANKJCO   SIMINC
blanking_file = blank.jcz       #BLANKFILE
si_file = siminc.dat            #SIFILE
1                               #SIMINCALC
.5  .5  5  1 95 1  #FRACOBS FRACPAR FULLJCOITN SIMINCACCEL MINSIMINC MATCHAGENTS
```

Figure 10.2. Example of a "simultaneous parameter increments" section of a PEST control file.

10.4.2 Variables

The role of each of the variables appearing in the "simultaneous parameter increments" section of a PEST control file is now discussed.

BLANKJCO

BLANKJCO is an integer variable which must be set to -1, 0 or 1.

If BLANKJCO is set to 1, PEST_HP reads a blanking file; it uses the contents of this file to blank every Jacobian matrix that it calculates. Note that Jacobian matrix blanking does not necessarily require that simultaneous parameter increments be employed for filling of the Jacobian matrix. As is stated above, Jacobian matrix blanking may raise the integrity of a Jacobian matrix calculated using random parameter increments.

Unless PEST_HP calculates its own blanking matrix (which occurs when the SIMINCCALC control variable is set to 1), the blanking file read by PEST_HP when BLANKJCO is set to 1 will normally have been written by the JCOBLANK utility. The name of this file is assigned to the BLANKFILE control variable. PEST_HP reads, and then re-reads, this file during every iteration of the inversion process just after it has undertaken the model runs required for filling of the Jacobian matrix.

As is usual practice, if PEST_HP is started with the "/i" command-line switch, it reads an existing Jacobian matrix for use in the first iteration of the inversion process. This matrix is <u>not</u> blanked, even if BLANKJCO is set to 1.

If BLANKJCO is set to -1, PEST_HP performs Jacobian blanking during every iteration of the inversion process. However under these circumstances SIMINC must be set to 0, as should the SIMINCCALC control variable (see below). Under these circumstances PEST_HP simply blanks every Jacobian matrix that it calculates using the FRACOBS and FRACPAR settings that are listed after the SIMINCCALC control variable. Under these circumstances, it neither reads nor records a blanking file; nor does it devise a simultaneous parameter increment strategy. Blanking based on FRACOBS and FRACPAR is applied to the Jacobian matrix which it has just calculated, on every occasion that it fills a Jacobian matrix. This BLANKJCO setting can sometimes be useful where randomized parameter increments are employed for filling of the Jacobian matrix. Note also that when BLANKJCO is set to -1, Jacobian blanking takes place during the first iteration of the inversion process, even if PEST_HP execution is initiated using the "/i" switch.

SIMINC

SIMINC, an integer variable, must be set to 0 or 1. If it is set to 1, PEST_HP obtains a simultaneous parameter increment strategy from a user-nominated simultaneous increment file; it employs this strategy during every iteration of the inversion process. The name of this file is assigned to the SIFILE control variable. Normally this file will have been written by the JCOBLANK utility. PEST_HP reads, and then re-reads, this file during every iteration of the inversion process just before it commissions the model runs required for filling of that iteration's Jacobian matrix.

If SIMINC (or SIMINCCALC) is set to 1, PEST_HP requires that the FORCEN variable for all parameter groups be set to the same value, this being either *always_2*, *always_3*, *switch*, or *always_5*. If different parameter groups have different FORCEN settings, the number of model runs used for calculation of derivatives of model outputs with respect to different parameters differs between parameters. This makes design of a simultaneous parameter increment strategy difficult. Actually, a FORCEN setting of *always_2* may be suitable for most occasions. While this setting precludes the attainment of increased derivatives precision through use of higher order finite differences, this may not matter too much as use of simultaneous parameter increments together with concomitant Jacobian element blanking also reduces the precision of finite-difference derivatives.

Note the following.

- If SIMINC is set to 1, PEST_HP will object if BLANKJCO is not also set to 1.
- If SIMINC is set to 1, PEST_HP will object if randomized Jacobian matrix calculation is activated.

BLANKFILE

BLANKFILE is the name of the Jacobian blanking file which PEST_HP reads if BLANKJCO is set to 1. As is illustrated in figure 10.2, its name must follow the string "blanking_file =" on the third line of the "simultaneous parameter increments" section of a PEST control file. Spaces can optionally surround the "=" sign. If BLANKJCO is set to 0, it is not necessary for the name of a file to follow the "blanking_file =" string, for its name is disregarded under these circumstances. If the name of the blanking file contains a space, its name should be enclosed by quotes.

The blanking file whose name is assigned to the BLANKFILE control variable must be a binary file; its name must have an extension of "*.jcz*". Normally this file will have been written by JCOBLANK. However it must not be named *case.jcz* where *case* is the filename base of the PEST control file, for this name is reserved for the blanking file that PEST_HP writes and reads if SIMINCCALC is set to 1.

SIFILE
SIFILE is the name of the simultaneous increment file which PEST_HP reads if SIMINC is set to 1. As is illustrated in figure 10.2, its name must follow the string "si_file =" on the fourth line of the "simultaneous parameter increments" section of a PEST control file. Spaces can optionally surround the "=" sign. If SIMINC is set to 0, it is not necessary for the name of a file to follow the "si_file =" string, for its name is disregarded under these circumstances. If the name of the simultaneous increments file contains a space, its name should be surrounded by quotes.

The simultaneous increment file whose name is supplied to the SIFILE control variable will normally have been written by JCOBLANK. This file must not be named *case.sif*, were *case* is the filename base of the PEST control file. This name is reserved for the simultaneous increment file which is written and read by PEST_HP if SIMINCCALC is set to 1.

SIMINCCALC
Set this integer variable to 1 to inform PEST_HP that it must periodically calculate a blanking matrix and devise a simultaneous parameter increment strategy itself. If PEST_HP devises a simultaneous parameter increment strategy during any particular iteration of the inversion process, it does so immediately after it has filled the Jacobian matrix pertaining to that iteration. First it determines a blanking matrix based on weighted sensitivities recorded in the Jacobian matrix that it has just calculated. This matrix is recorded in a file named *case.jcz*, where *case* is the filename base of the PEST control file. Then, based on this blanking matrix, PEST_HP devises a simultaneous parameter increment strategy for use during one or more subsequent iterations. It stores this strategy in a simultaneous increment file named *case.sif*, where *case* is the filename base of the PEST control file.

Note that just after PEST_HP has devised a simultaneous parameter increment strategy, it re-examines the blanking matrix on which basis this strategy was devised. If it is possible to unblank some elements of this matrix while maintaining the integrity of the simultaneous parameter increment strategy, then it does so. As was discussed above, this can increase the precision of (simultaneous) finite-difference derivatives calculation.

If SIMINCCALC is set to 1, PEST_HP does not use a PEST_HP-calculated simultaneous parameter increment strategy to fill the Jacobian matrix that it calculates during the first iteration of the inversion process. If SIMINC is set to 0, this matrix is filled in the normal way; that is, all parameters are incremented individually. Alternatively, if SIMINC is set to 1, the Jacobian matrix used in the first iteration is filled using a simultaneous parameter increment strategy supplied in a file whose name is ascribed to the SIFILE control variable; the Jacobian matrix is then blanked using the blanking matrix supplied in a file whose name is ascribed to the BLANKFILE control variable. The Jacobian matrix computed during all iterations identified by the value of the FULLJCOITN control variable (see below) is computed in this same way (if SIMINC is set to 1). During all of these iterations, however, PEST_HP evaluates its own blanking matrix and simultaneous parameter increment strategy based on the thus-computed Jacobian matrix if SIMINCCALC is set to 1. These are stored in files *case.jcz* and *case.sif* for use in subsequent iterations; existing versions of these files are thus over-written.

If SIMINCCALC is set to 1, then PEST_HP will object if its randomized Jacobian functionality is activated.

FRACOBS and FRACPAR

PEST_HP calculates a blanking matrix from a Jacobian matrix in the same way that JCOBLANK does. (Unlike JCOBLANK, however, PEST_HP does not optionally read a zeroing file to obtain blanking information.) First PEST_HP blanks all rows of the Jacobian matrix for which observation weights are zero. Then, for each row, it ascertains the weighted sensitivity of highest absolute value. Any elements within this row whose absolute weighted sensitivities are less than FRACOBS times this maximum value are then blanked. The same operation is then performed for each column of the weighted Jacobian matrix using FRACPAR as the governing variable. FRACOBS and FRACPAR should both lie between 1.0 and 0.0 (inclusive); they are real variables.

Note that prior information sensitivities are not blanked. Note also that the blanking matrix and simultaneous parameter increment strategy devised by PEST_HP during any iteration of the inversion process are not used until the ensuing iteration.

FULLJCOITN

As stated above, if SIMINCCALC is set to 1, PEST_HP calculates a Jacobian matrix in the “normal” way during the first iteration of the inversion process. Hence it does not employ simultaneous parameter increments for filling of this matrix unless SIMINC is set to 1. On the basis of this Jacobian matrix it calculates a blanking matrix and devises a simultaneous parameter increment strategy for use in the next iteration (or more) of the inversion process.

PEST_HP also calculates a Jacobian matrix in the normal way at intervals of FULLJCOITN iterations. Thus, for example, if FULLJCOITN is set to 2, then PEST_HP calculates a Jacobian matrix in the normal manner during iterations 1, 3, 5, 7, etc of the inversion process; similarly, if FULLJCOITN is set to 3, PEST_HP calculates a Jacobian matrix in the normal way during iterations 1, 4, 7, 10, etc of the inversion process. During all other iterations of the inversion process, PEST_HP employs simultaneous increment and blanking strategies which it has calculated itself; it reads the blanking matrix and details of the simultaneous parameter increment strategy from files *case.jcz* and *case.sif* respectively, where *case* is the filename base of the PEST control file.

PEST_HP will cease execution with an error message if FULLJCOITN is set to 1 or less.

SIMINCACCEL

“SIMINCACCEL” stands for “simultaneous increment acceleration”. If SIMINCACCEL is set to 1 then SIMINCCALC should also be set to 1.

If SIMINCACCEL is set to 0 and SIMINCCALC is set to 1, PEST_HP does not compute a blanking matrix or simultaneous parameter increment strategy during those iterations in which it actually uses a blanking matrix and simultaneous parameter increment strategy that it has previously calculated itself. This happens during all iterations except those for which the setting of the FULLJCOITN control variable dictates that Jacobian matrix calculation takes place in the "normal" way. However if SIMINCACCEL is set to 1, then PEST_HP updates the blanking matrix and evaluates a new simultaneous parameter increment strategy even during those iterations in which it is using a blanking matrix and simultaneous parameter increment strategy that it has previously calculated for itself. This can result in a significant reduction in the number of model runs required for filling of the Jacobian matrix during the ensuing iteration (as long as this iteration has not been decreed by FULLJCOITN to be a "normal" Jacobian iteration). Unfortunately, this reduction in model run requirements may be accompanied by some loss of integrity of the Jacobian matrix that is calculated during that iteration. Hence if SIMINCACCEL is set to 1, it is good practice to set the MINSIMINC control variable (see below) to a sensible value so that the number of model runs employed for filling the Jacobian matrix does not become unworkably low.

If SIMINCACCEL is set to 1 then FULLJCOITN must be set to 3 or greater. If FULLJCOITN is not set to 3 or greater under these circumstances, then there can be no iterations to which a non-zero value of SIMINCACCEL actually pertains. This is because PEST_HP is either calculating a Jacobian matrix in the "normal" way during a particular iteration, or is employing a blanking strategy that it has devised itself on the basis of this "normally-calculated" Jacobian matrix.

MINSIMINC
If this integer variable is set to a number greater than 0, PEST_HP will never devise a parameter increment strategy which lowers the number of simultaneous parameter increments below this number.

If the value supplied for MINSIMINC is equal to or greater than the number of adjustable parameters, PEST_HP will cease execution with an error message.

MATCHAGENTS
If SIMINCCALC is set to 1, so that PEST_HP devises its own simultaneous parameter increment strategy during some iterations of the inversion process, PEST_HP will try to ensure that the number of simultaneous parameter increments that it adopts is an integral multiple of the number of agents available for carrying out model runs. First PEST_HP evaluates a simultaneous parameter increment strategy using the FRACOBS and FRACPAR control variables described above. Then, if necessary, it raises the number of simultaneous parameter increments calculated in this way until the number of increments is equal to a multiple of the number of available agents. This ensures that no agents are idle as the Jacobian matrix is being filled. Raising the number of simultaneous increments to achieve processor-to-increment parity also raises the integrity of the thus-calculated Jacobian matrix, as less blanking of this matrix is required to forestall the assignment of changes in model outputs to the wrong parameters.

10.5 Using SVD-Assist

PEST_HP allows the use of simultaneous parameter increments when undertaking SVD-assisted parameter estimation. However because the use of super parameters already constitutes a theoretically optimal run reduction strategy, little is likely to be gained by using both methods together.

The SVDAPREP utility which prepares a super-parameter PEST control file tolerates the presence of a "simultaneous parameter increments" section in the base parameter PEST control file from which the super parameter PEST control file is derived. However it does not transfer this section to the super parameter PEST control file. It is the user’s responsibility to add this section to the super parameter PEST control file him/herself.

11. Null Space Monte Carlo

PEST’s null space Monte Carlo (NSMC) functionality is described in PEST documentation and in the PEST book. Briefly, many random parameter sets are generated which “almost calibrate” a model. (The RANDPAR, RANDPAR1, PNULPAR, and possibly PREDUNC7 utilities supplied with the normal version of PEST can be used in preparing these parameter sets). Then, by repeatedly running PEST using its SVD-assist functionality, all of these parameter sets can be adjusted with minimal computational burden so that they calibrate the model to within a user-specified tolerance. (That is, they can all be adjusted so that the objective function associated with each one of them is reduced below a user-specified threshold.)

There is no reason why PEST_HP cannot be used to perform repeated calibration exercises of this type. To automate this process, two batch files can be used – one to initiate repeated runs of the PEST_HP manager, and the other to initiate repeated runs of each of the AGENT_HP agents. The first of these batch files should be run from PEST_HP’s working folder. Meanwhile, a copy of the second of these batch files should be placed in the working folder of each agent and run from there.

Figure 11.1 shows an example of a batch file used for running PEST_HP.

```
rem ############################################################
rem Delete an existing record file.
rem ############################################################

del /P record.dat
echo  > record.dat
pause

rem ############################################################
rem Do all the PEST_HP runs.
rem ############################################################

for /L %%i in (1,1,50) do (
parrep nrandom%%i.par base.pst.kp1 base.pst
pest_hp base_svda /h :4004
find /I "ie phi" base_svda.rec >> record.dat
copy base.bpa base.bpa.%%i
copy base_svda.rec base_svda.rec.%%i
timeout /t 2
)
```

Figure 11.1 A batch file used to initiate repeated runs of PEST_HP when implementing PEST’s NSMC methodology.

In the example of figure 11.1 a PEST control file named *base.pst* holds base parameters. Randomly generated, null-space projected, parameter sets are housed in parameter value files named *nrandom*.par* where “*” is replaced by 1, 2, 3 etc., these constituting parameter set indices. Random parameter sets are sequentially introduced to the base PEST control file as initial values in this file using the PARREP utility. The super parameter PEST control file is named *base_svda.pst*. The script provided in figure 11.1 runs PEST_HP repeatedly in order to adjust these parameter sets so that each of them fits the calibration dataset better. Normally the NOPTMAX variable in file *base_svda.pst* should be set to 1, or perhaps 2, to place an upper limit on the number of model runs that are committed to this process. After each PEST_HP run is complete, the parameter value file containing adjusted parameters, and the corresponding PEST_HP run record file which documents the inversion process, are copied to files *base.bpa.i* and *base_svda.rec.i* respectively, where *i* is the parameter set index.

The batch file illustrated in figure 11.2 can be placed in each of the agent folders.

```
:label1
agent_hp base_svda /h hostname:4004
timeout /t 2
goto label1
```

Figure 11.2 A batch file used to initiate repeated runs of AGENT_HP.

The batch file depicted in figure 11.2 needs to be run only one; replace *hostname* in this file with the IP address or name of the computer on which the PEST_HP manager is running. On completion of each inversion process, execution of AGENT_HP is re-initiated in readiness for the next inversion process.

Notice the line:

```
timeout /t 2
```

in each of figures 11.1 and 11.2. This WINDOWS system command creates a pause of 2 seconds in the batch processing sequence. This is not essential, but allows a few moments to elapse for manager-agent messaging to catch up with the batch processing sequence; this pause may need to be longer where there are many agents and where network traffic is high.

It is often a good idea to implement Broyden Jacobian updating when using PEST_HP in the NSMC process. As is described in PEST documentation, one of the means through which the NSMC process achieves a high degree of model-run efficiency is through re-use of the same Jacobian matrix for the first iteration of each random parameter set adjustment process. This matrix is usually calculated on the basis of the calibrated parameter set rather than on the basis of any one random parameter set. Parameter-set-specific improvements in this matrix achieved through Broyden updating may allow the first iteration of each random parameter set adjustment process to achieve more than it otherwise would in terms of reducing the objective function.

12. Secondary Parameters

12.1 General

Secondary parameters can appear in a PEST control file that is read by any member of the HP suite of programs, including PEST_HP. The use of secondary parameters removes the need to employ PAR2PAR in a batch or script file that is run as "the model".

Secondary parameters are not optimised. Their values are simply calculated from primary (i.e. normal) parameters using equations supplied in the PEST control file. These equations can feature the primary parameters that are cited in a PEST control file (including fixed and tied parameters) and other secondary parameters whose values were previously calculated. They can also feature so-called "file parameters". See the "HP Suite" companion to this manual for a description of these parameter types. PEST_HP does not support the use of file parameters; however CMAES_HP, a member of the HP suite, does support their use.

12.2 Defining Secondary Parameters

A secondary parameter can be defined in a PEST control file anywhere within the "parameter data" section of this file. Figure 12.1 shows an example of the "parameter data" section of a PEST control file in which secondary parameters are defined.

```
* parameter data
ro1  log factor 1.000000 0.1 100 ro 1 0 1
ro2  log factor 1.000000 0.1 100 ro 1 0 1
ro3  log factor 1.000000 0.1 100 ro 1 0 1
ro4  log factor 1.000000 0.1 100 ro 1 0 1
ro5  log factor 1.000000 0.1 100 ro 1 0 1
ro6  log factor 1.000000 0.1 100 ro 1 0 1
ro7  log factor 1.000000 0.1 100 ro 1 0 1
ro8  log factor 1.000000 0.1 100 ro 1 0 1
ro9  log factor 1.000000 0.1 100 ro 1 0 1
ro10 log factor 1.000000 0.1 100 ro 1 0 1
h1   log factor  0.25  0.05 100 hhh 1 0 1
h2   tied factor  0.50 0.05 100 hhh 1 0 1
h3   tied factor  1.00 0.05 100 hhh 1 0 1
h4   tied factor  2.00 0.05 100 hhh 1 0 1
h5   tied factor  4.00 0.05 100 hhh 1 0 1
h6   tied factor  8.00 0.05 100 hhh 1 0 1
ep1 = ro1+ro2
ep2 = ep1+ep1
h7   tied factor  16.0 0.05 100 hhh 1 0 1
h8   tied factor  32.0 0.05 100 hhh 1 0 1
h9   tied factor  64.0 0.05 100 hhh 1 0 1
h2 h1
h3 h1
ep2 = 2*ep2
h4 h1
h5 h1
h6 h1
h7 h1
h8 h1
h9 h1
```

Figure 12.1 The "parameter data" section of a PEST control file wherein secondary parameters are defined.

Secondary parameters are defined by an equation. Their presence on any line of the "parameter data" section of a PEST control file is recognized by the fact that an "=" symbol follows their name.

An equation must follow the "=" symbol. This equation can be of arbitrary complexity; see documentation of PAR2PAR and PLPROC (both of which feature parameter equations) for examples. The right side of the equation can feature any primary PEST parameter, file parameter, or previously-defined secondary parameter. As in any programming language, a secondary parameter can appear on both sides of an equation, provided a value has already been calculated for it; its value can thus be updated by the equation.

If desired, the equation through which a secondary parameter is given a value can have a logical outcome and involve logical operators; see "selection equations" in documentation of the PLPROC program. If the outcome of an equation is logical, then a calculated value of TRUE endows the secondary parameter with a value of 1.0, while a calculated value of FALSE endows the secondary parameter with a value of 0.0.

The values calculated for secondary parameters are transferred from the PEST_HP manager to its agents at the same time, and in the same manner, as are the values of primary parameters. These values are then transferred to model input files using template files. Hence template files which are listed in the "model input/output" section of a PEST control file in which secondary parameters are defined can also feature secondary parameters. Note that it is not necessary for all secondary parameters defined in a PEST control file to appear in one or more template files which are cited in that file. Some secondary parameters may thus be used only for intermediate calculations whose ultimate goal is the assignment of a value to another secondary parameter whose value is then written to a model input file.

12.3 Number of Secondary Parameters and Equations

PEST_HP (and other HP suite programs) must be informed of the existence of secondary parameters, and of equations which define them, in the "control data" section of the PEST control file which they read. This information is used to dimension arrays which hold data pertaining to these entities. See figure 12.2.

```
pcf
* control data
restart estimation
10 19 2 10 3 nparsec=2 nequation=3
3 3 single point  1 0 0
Etc
```

Figure 12.2 First part of the "control data" section of a PEST control file which features secondary parameters.

On the fourth line of the PEST control file depicted in figure 12.2, the string "nparsec=2" informs PEST_HP that there are 2 secondary parameters. The string "nequations=3" informs PEST_HP that there are 3 equations. (A space can precede or follow the "=" symbol in each case.) As is obvious from the above discussion, the existence of secondary parameters implies the existence of equations, and vice versa. Also, the number of equations must equal or exceed the number of secondary parameters. It is important to note that the NPAR variable (first variable featured on the fourth line of the PEST control file) must refer only to the number of primary PEST parameters. (Note also that, as well as featuring the NPARSEC and NEQUATION control variables, the fourth line of a PEST control file may also feature the NPARFILE and FILEPARFILE variables. These variables pertain to file parameters. As stated above, at the time of writing, only CMAES_HP can employ file-parameters.)

The following should also be noted.

- PEST_HP will cease execution with an appropriate error message if secondary parameters are defined in a PEST control file that asks it to undertake SVD-assisted inversion.
- Prior information equations cannot cite secondary parameters.
- The values of secondary parameters are not listed in parameter value files recorded by PEST_HP in which progressively optimised values of primary parameters are recorded. As secondary parameters are not optimised, but are functions of primary parameters, there is no need to record their values in these files.
- The values of secondary parameters associated with optimised primary parameters are listed at the end of the PEST_HP run record file.
- A primary parameter cannot appear on the left side of an equation.

12.4 Operation of PEST_HP with Secondary Parameters

Outwardly, the operation of PEST_HP when using secondary parameters is no different from its operation without them. The values of secondary parameters are calculated by the run manager prior to these values being transferred to run agents for transfer to model input files. If any problems are encountered in parsing the equation through which a secondary parameter is assigned its value, or with calculating the value of a secondary parameter, an error message is recorded and PEST_HP ceases execution.

The following features of PEST_HP operations when using secondary parameters should also be noted.

- If PEST_HP is run with the "/f" switch, it will accept secondary parameters in the PEST control file. However it does not record the values of secondary parameters in the run results file. (As stated above, even though these are featured in the PEST control file, they are not adjustable; hence they can be considered to be the outcome of intermediate calculations undertaken by the model.)
- Only PEST_HP, but not PEST, can parse and evaluate equations. Therefore PEST cannot be used to write a hp starter file for the use of PEST_HP if a PEST control file features secondary parameters.

13. Compatibility Issues

13.1 General

As has already been discussed, a control file for PEST_HP (and other HP suite programs) can cite a number of control variables that are not useable by standard versions of PEST. These are:

- RUN_ABANDON_FAC,
- WIN_MRUN_HOURS,
- SOFTSTOPHOURS and HARDSTOPHOURS,
- RRFSAVE,
- ZEROSENVAL,
- alternative LSQR control settings,
- UPTESTMIN and UPTESTLIM,
- ORR_NOT_FIRST,
- REG2MEASRAT,
- JCOWARNTHRESH and JCOZEROTHRESH.

A PEST_HP control file can also site secondary parameters.

The normal version of PEST is accompanied by a suite of utility programs that implement pre- and post-processing functionality of various types. Many of them read a PEST control file. At the time of writing, not all of these utility programs have been altered to recognize new variables and parameter types that are supported by members of the HP suite. Hence, if some of them are asked to read a PEST control file that contains variables and parameter types that are specific to the HP-suite, they may complain that the PEST control file contains an error, and/or that it should be checked with PESTCHEK. Some may offer a more obscure error message.

Eventually, all PEST-support programs will be altered to accommodate the presence of HP-specific variables and parameter types in a PEST control file. For the moment, however, use of utilities which do not accept these variables and parameter types requires that these variables and parameter types be removed from the PEST control file before these utilities are run. This can be accomplished automatically using the PSTCLEAN utility that is supplied with the PEST suite.

PEST-support utilities that have, at the time of writing, been modified to recognize HP-specific variables, and take appropriate action, include the following.

- Linear analysis utilities such as IDENTPAR and members of the PREDVAR* and PREDUNC* suites can happily read a PEST_HP control file. Where secondary parameters appear in a PEST control file that is read by these programs, they are ignored. This is because calculation of parameter and predictive error and uncertainty variance relies only on calibration-adjustable PEST parameters, and the sensitivities of model outputs to these parameters as recorded in a JCO file.
- The GENLINPRED program that runs many of these uncertainty and error analysis utilities can also accommodate a PEST_HP control file.

- PARREP reads a parameter value file and a PEST_HP control file. It creates a new PEST_HP control file in which the initial values of parameters are replaced by those recorded in the parameter value file. Lines in an existing PEST_HP control file which define secondary parameters and file-parameters are transferred directly to the new PEST_HP control file which PARREP writes.
- SVDAPREP will transfer the values of HP-specific control variables to the super parameter PEST_HP control file which it creates. However it will not write a super parameter PEST_HP control file if the base parameter PEST_HP control file features secondary parameters. Instead, it will cease execution with an appropriate error message. (It would make no sense to base secondary parameters on super parameters in a PEST_HP control file.)
- The RANDPAR* suite of utilities can read a PEST_HP control file that contains any of the HP-specific functionality discussed above. They can write a sequence of parameter value files which contain random values of primary (i.e. normal) PEST parameters that are featured in the control file. These utilities do not generate random values for secondary parameters; because secondary parameters are a derived parameter type, there is no need to cite them in parameter value files.
- JCO2JCO reads a PEST_HP control file and corresponding Jacobian matrix file. It can record a JCO file corresponding to another PEST_HP control file, notwithstanding the fact that either or both of these PEST_HP control files may contain HP-specific control variables and/or secondary (or file) parameters. Of course, a JCO file does not record sensitivities of model outputs to secondary (or file) parameters.
- JCOCHEK can check for compatibility between a PEST_HP control file and a JCO file, notwithstanding the presence of HP-specific control variables and/or file/secondary parameters in the PEST_HP control file.
- RRFCALCPSI can evaluate objective functions from parameter and model output values recorded in a run results file, regardless of the presence or otherwise of file/secondary parameters in the PEST_HP control file which it reads.
- ADDREG1, ADDREG2 and SUBREG1 can add/subtract regularisation to/from a PEST control file containing HP-specific control variables and/or file/secondary parameters.
- WTFACTOR can also accommodate all nuances of a PEST_HP control file.
- PESTCHEK can read and check the contents of a PEST_HP input dataset. This dataset can include HP-specific control variables, secondary parameters and file parameters. PESTCHEK issues a warning message where it encounters functionality which is specific only to programs of the HP suite.
- PEST and BEOPEST can read a PEST control file which features variables and parameters that are specific to the HP suite. They will then cease execution with an appropriate error message, drawing the user's attention to functionality that they are incapable of supporting. If PEST is started with the "/hpstart" switch, however, it will not complain about the presence of HP-specific control variables in a PEST control file; instead it will undertake a single model run and record a hp starter file. It will not do this, however, if a PEST control file features secondary parameters or file parameters, for it has not been programmed to process these types of parameters. Instead it ceases execution with an appropriate message.

13.2 The PSTCLEAN Utility

The PSTCLEAN utility is a member of the PEST suite. It reads a PEST control file which can serve as a suitable input file for PEST_HP, or any other member of the HP suite. It writes a new PEST control file from which all HP-specific variables have been removed. The new PEST control file is readable by all PEST utility software.

As is explained in Part II of the PEST manual, PSTCLEAN also removes variables which are specific to PEST++, as well as comments, from a PEST control file. PEST++ variables are placed on lines that begin with the “++” string. Comments can be placed on any line of a PEST control file following a “#” character. PEST_HP tolerates both of these; hence there is no need to remove “++” lines or comments from a PEST control file before using PEST_HP or any other member of the HP suite.

14. References

Chada, N.K., Iglesias, M.A., Roininen, L. and Stuart, A.M., 2018. Parameterizations for ensemble Kalman inversion. *Inverse Problems* 34, 055009 (3100)

Chen, T. and Oliver, D.S., 2013. Levenberg-Marquardt foms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Computational Geosciences*, 17 (4), 689-703.

Engel, J., Buydens, L. and Blanchet, L., 2017. An overview of large-dimensional covariance and precision matrix estimators with applications in chemometrics. *Journal of Chemometrics*. DOI 10.1002/cem.2880.

Halko, N., Martinsson, P.G. and Tropp, J.A., 2011. Finding structure in randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*. 53 (2), 217-288.

Luo, X. and Bhakta, T., 2020. Automatic and adaptive localization for ensemble-based history-matching. *Journal of Petroleum Science Engineering*. 184, 106559.

Tropp, J.A., Yurtsever, A, Udell, M. and Cevher, V., 2016. Randomized single-view algorithms for low-rank matrix approximation. *arXiv preprint arXiv:1609000481208*

White, J.T., 2018. A model-independent iterative ensemble smoother for efficient history matching and uncertainty quantification in very high dimensions. *Environmental Modeling and Software*, 109. doi: 10.1016/j.envsoft.2018.06.009.

Appendix 1: New Control Variables Illustrated

Figure A1.1 shows the “control data” section of a PEST control file wherein values are specified for the UPTESTMIN, UPTESTLIM, RUN_SLOW_FAC, RUN_ABANDON_FAC, WIN_MRUN_HOURS, SOFTSTOPHOURS, ZEROSENVAL, JCOWARNTHRESH and JCOZEROTHRESH variables. All of these variables are optional; hence they can be omitted from a PEST_HP control file if desired. As has been discussed, a value cannot be supplied for both of SOFTSTOPHOURS and HARDSTOPHOURS in the same PEST control file. The protocol for supplying a value for HARDSTOPHOURS is identical to that for SOFTSTOPHOURS; simply replace the “softstophours” string with “hardstophours” in the example below.

Use of the OBSREREF, ORR_NOT_FIRST and RRFSAVE variables is also demonstrated in figure A1.1.

```
* control data
restart regularisation
19 19 2 10 3
2 3 single point 1 0 0 obsreref orr_not_first
.1 2 0.3 0.03 3 3 uptestmin=10 uptestlim=50 derforgive run_slow_fac=2.5 run_abandon_fac=5.0 win_mrun_hours=0.5
2 3 0.001 0
0.5  zerosenval=-1.11E29 jcowarnthresh=1.0e20 jcozerothresh=1.0e30
30 0.01 3 3 0.01 3 softstophours=24.0
1 1 1 rrfsave
```

Figure A1.1 Example of the “control data” section of a PEST_HP control file.