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 **J** is the Jacobian matrix and **Q** is the weight matrix. **Q** is often diagonal, with elements equal to the squares of user-supplied weights. However **J** can also include Tikhonov regularisation which may be accompanied by one or more covariance matrices. In that case **Q** includes the inverse of these matrices as well.

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

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

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

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

and λ is the value of the Marquardt lambda. The "1" subscript on **V** and **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

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

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

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

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, $\delta\mathbf{k}$ 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 “>@”.