Addendum
to the
PEST Manual

John Doherty
Watermark Numerical Computing
May 2013

Acknowledgements

I would like to acknowledge the help of the following institutions who provided me with some much needed assistence in developing the software and methodologies documented herein.

- The National Centre for Groundwater Research and Training, Australia.
- S. S. Papadopulous and Associates, USA.
- Environmental Simulations Inc., USA.
- Aquaveo USA.
- United States Geological Survey.
- Matrix Solutions, Canada.

Preface: The PEST Control File

For ease of reference, variables within the PEST control file are listed, together with a short explanation of each. This information is the most recent, and includes all variables discussed in both this addendum and in the PEST manual itself.

```
pcf
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM]
NTPLFLE NINSFLE PRECIS DPOINT [NUMCOM JACFILE MESSFILE] [OBSREREF]
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE] [LAMFORGIVE] [DERFORGIVE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND] [ABSPARMAX]
PHIREDSWH [NOPTSWITCH] [SPLITSWH] [DOAUI] [DOSENREUSE] [BOUNDSCALE]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN] [PHIABANDON]
ICOV ICOR IEIG [IRES] [JCOSAVE] [VERBOSEREC] [JCOSAVEITN] [REISAVEITN] [PARSAVEITN] [PARSAVERUN]
* automatic user intervention
MAXAUI AUISTARTOPT NOAUIPHIRAT AUIRESTITN
AUISENSRAT AUIHOLDMAXCHG AUINUMFREE
AUIPHIRATSUF AUIPHIRATACCEPT NAUINOACCEPT
* singular value decomposition
SVDMODE
MAXSING EIGTHRESH
EIGWRITE
* lsqr
LSQRMODE
LSQR_ATOL LSQR_BTOL LSQR_CONLIM LSQR_ITNLIM
LSQRWRITE
* svd assist
BASEPESTFILE
BASEJACFILE
SVDA_MULBPA SVDA_SCALADJ SVDA_EXTSUPER SVDA_SUPDERCALC SVDA_PAR_EXCL
* sensitivity reuse
SENRELTHRESH  SENMAXREUSE
SENALLCALCINT  SENPREDWEIGHT  SENPIEXCLUDE
* parameter groups
PARGPNME INCTYP DERINC DERINCLB FORCEN DERINCMUL DERMTHD [SPLITTHRESH SPLITRELDIFF SPLITACTION]
(one such line for each of NPARGP parameter groups)
* parameter data
PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM
(one such line for each of NPAR parameters)
PARNME PARTIED
(one such line for each tied parameter)
* observation groups
OBGNME [GTARG] [COVFLE]
(one such line for each of NOBSGP observation group)
* observation data
OBSNME OBSVAL WEIGHT OBGNME
(one such line for each of NOBS observations)
* derivatives command line
DERCOMLINE
EXTDERFLE
* model command line
COMLINE
(one such line for each of NUMCOM command lines)
* model input/output
TEMPFLE INFLE
(one such line for each of NTPLFLE template files)
INSFLE OUTFLE
(one such line for each of NINSLFE instruction files)
* prior information
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of NPRIOR articles of prior information)
* predictive analysis
NPREDMAXMIN [PREDNOISE]
PD0 PD1 PD2
ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH
ABSPREDSWH RELPREDSWH
NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP
* regularisation
```

```
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE]
WFINIT  WFMIN  WFMAX  [LINREG] [REGCONTINUE]
WFFAC  WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]
* pareto
PARETO_OBSGROUP
PARETO_WTFAC_START PARETO_WTFAC_FIN NUM_WTFAC_INC
NUM_ITER_START NUM_ITER_GEN NUM_ITER_FIN
ALT_TERM
OBS_TERM ABOVE_OR_BELOW OBS_THRESH NUM_ITER_THRESH (only if ALT_TERM is non-zero)
NOBS_REPORT
OBS_REPORT_1 OBS_REPORT_2 OBS_REPORT_3.. (NOBS_REPORT items)
```

Variables in "control data" section of PEST control file.

Variable	Type	Values	Description
RSTFLE	text	"restart" or "norestart"	instructs PEST whether to write restart data
PESTMODE	text	"estimation", "prediction", "regularisation", "pareto"	PEST's mode of operation
NPAR	integer	greater than zero	number of parameters
NOBS	integer	greater than zero	number of observations
NPARGP	integer	greater than zero	number of parameter groups
NPRIOR	integer	any integer value	absolute value is number of prior information equations; negative value indicates supply of prior information in indexed format
NOBSGP	integer	greater than zero	number of observation groups
MAXCOMPDIM	integer	optional; zero or greater	number of elements in compressed Jacobian matrix
NTPLFLE	integer	greater than zero	number of template files
NINSFLE	integer	greater than zero	number of instruction files
PRECIS	text	"single" or "double"	format for writing parameter values to model input files
DPOINT	text	"point" or "nopoint"	omit decimal point in parameter values if possible
NUMCOM	integer	optional; greater than zero	number of command lines used to run model
JACFILE	integer	optional; 0, 1 or -1	indicates whether model provides external derivatives file
MESSFILE	integer	optional; zero or one	indicates whether PEST writes PEST-to-model message file
OBSREREF	text	"obseref" or "noobsreref"	activates or de-activates observation re-referencing
RLAMBDA1	real	zero or greater	initial Marquardt lambda
RLAMFAC	real	positive or negative, but not zero	dictates Marquardt lambda adjustment process
PHIRATSUF	real	between zero and one	fractional objective function sufficient for end of current iteration
PHIREDLAM	real	between zero and one	termination criterion for Marquardt lambda search
NUMLAM	integer	one or greater; possibly negative with Parallel or BEOPEST	maximum number of Marquardt lambdas to test

JACUPDATE	integer	optional; zero or greater	activation of Broyden's Jacobian update procedure
LAMFORGIVE	text	"lamforgive" or "nolamforgive"	treat model run failure during lambda search as high objective function
DERFORGIVE	text	"derforgive" or "noderforgive"	accomodates total or partial model failure during Jacobian runs by setting pertinent sensitivities to zero
RELPARMAX	real	greater than zero	parameter relative change limit
FACPARMAX	real	greater than one	parameter factor change limit
FACORIG	real	between zero and one	minimum fraction of original parameter value in evaluating relative change
ABSPARMAX	real	greater than zero	parameter absolute change limit
IBOUNDSTICK	integer	optional; zero or greater	instructs PEST not to compute derivatives for parameter at its bounds
UPVECBEND	integer	optional; zero or one	instructs PEST to bend parameter upgrade vector if parameter hits bounds
PHIREDSWH	real	between zero and one	sets objective function change for introduction of central derivatives
NOPTSWITCH	integer	optional; one or greater	iteration before which PEST will not switch to central derivatives computation
SPLITSWH	real	optional; zero or greater	the factor by which the objective function rises to invoke split slope derivatives analysis until end of run
DOAUI	text	"aui", "auid", or "noaui"	instructs PEST to implement automatic user intervention
DOSENREUSE	text	"senreuse" or "nosenreuse"	instructs PEST to re-use parameter sensitivities
BOUNDSCALE	text	"boundscale" or "noboundscale"	Parameters are scaled by the inter-bounds interval if using singular value decomposition, LSQR or SVDA.
NOPTMAX	integer	-2, -1, 0, or any number greater than zero	number of optimisation iterations
PHIREDSTP	real	greater than zero	relative objective function reduction triggering termination
NPHISTP	integer	greater than zero	number of successive iterations over which PHIREDSTP applies
NPHINORED	integer	greater than zero	number of iterations since last drop in objective function to trigger termination
RELPARSTP	real	greater than zero	maximum relative parameter change triggering termination

NRELPAR	integer	greater than zero	number of successive iterations over which RELPARSTP applies
PHISTOPTHRESH	real	optional; zero or greater	objective function threshold triggering termination
LASTRUN	integer	optional; zero or one	instructs PEST to undertake (or not) final model run with best parameters
PHIABANDON	real or text	optional	objective function value at which to abandon optimisation process or filename containing abandonment schedule
ICOV	integer	zero or one	record covariance matrix in matrix file
ICOR	integer	zero or one	record correlation coefficient matrix in matrix file
IEIG	integer	zero or one	record eigenvectors in matrix file
IRES	integer	zero or one	record resolution data
JCOSAVE	text	"jcosave" or "nojcosave"	Save best Jacobian file as a JCO file - overwriting previously-saved files of the same name as the inversion process progresses.
VERBOSEREC	text	"verboserec" or "noverboserec"	If set to "noverboserec", parameter and observation data lists are ommitted from the run record file.
JCOSAVEITN	text	"jcosaveitn" or "nojcosaveitn"	Write current jacobian matrix to iteration-specific JCO file at the end of every optimisation iteration.
REISAVEITN	text	"reisaveitn" or "noreisaveitn"	Store best-fit residuals to iteration-specific residuals file at end of every optimisation iteration.
PARSAVEITN	text	"parsaveitn" or "noparsaveitn"	Store iteration specific parameter value files.
PARSAVERUN	text	"parsaverun" or "noparsaverun"	Store run specific parameter value files.

Variables in optional "automatic user intervention" section of PEST control file.

Variable	Type	Values	Description
MAXAUI	integer	zero or greater	maximum number of AUI iterations per optimisation iteration
AUISTARTOPT	integer	one or greater	optimisation iteration at which to commence AUI
NOAUIPHIRAT	real	between zero and one	relative objective function reduction threshold triggering AUI
AUIRESTITN	integer	zero or greater, but not one	AUI rest interval expressed in optimisation iterations
AUISENSRAT	real	greater than one	composite parameter sensitivity ratio triggering AUI
AUIHOLDMAXCHG	integer	zero or one	instructs PEST to target parameters which change most when deciding which parameters to hold
AUINUMFREE	integer	greater than zero	cease AUI when only AUINUMFREE parameters are unheld
AUIPHIRATSUF	real	between zero and one	relative objective function improvement for termination of AUI
AUIPHIRATACCEPT	real	between zero and one	relative objective function reduction threshold for acceptance of AUI-calculated parameters
NAUINOACCEPT	integer	greater than zero	number of iterations since acceptance of parameter change for termination of AUI

Variables in optional "singular value decomposition" section of PEST control file.

Variable	Type	Values	Description
SVDMODE	integer	zero or one	activates truncated singular value decomposition for solution of inverse problem
MAXSING	integer	greater than zero	number of singular values at which truncation occurs
EIGTHRESH	real	zero or greater, but less than one	eigenvalue ratio threshold for truncation
EIGWRITE	integer	zero or one	determines content of SVD output file

Variables in optional "LSQR" section of PEST control file.

Variable	Type	Values	Description
LSQRMODE	integer	zero or one	activates LSQR solution of inverse problem
LSQR_ATOL	real	zero or greater	LSQR algorithm *atol* variable
LSQR_BTOL	real	zero or greater	LSQR algorithm *btol* variable
LSQR_CONLIM	real	zero or greater	LSQR algorithm *conlim* variable
LSQR_ITNLIM	integer	greater than zero	LSQR algorithm *itnlim* variable
LSQR_WRITE	integer	zero or one	instructs PEST to write LSQR file

Variables in optional "SVD-assist" section of PEST control file.

Variable	Type	Values	Description
BASEPESTFILE	text	a filename	name of base PEST control file
BASEJACFILE	text	a filename	name of base PEST Jacobian matrix file
SVDA_MULBPA	integer	zero or one	instructs PEST to record multiple BPA files
SVDA_SCALADJ	integer	-4 to 4	sets type of parameter scaling undertaken in super parameter definition
SVDA_EXTSUPER	integer	0, 1, 2, -2, 3	sets means by which super parameters are calculated
SVDA_SUPDERCALC	integer	zero or one	instructs PEST to compute super parameter sensitivities from base parameter sensitivities
SVDA_PAR_EXCL	integer	0, 1 or -1	if set to 1, instructs PEST to compute super parameters on basis only of observation group in base parameter PEST control file to which pareto-adjustable weighting is assigned in super parameter PEST control file. If set to -1 all groups other than this form basis for super parameter definition.

Variables in optional "sensitivity re-use" section of PEST control file.

Variable	Type	Values	Description
SENRELTHRESH	real	zero to one	relative parameter sensitivity below which sensitivity re-use is activated for a parameter
SENMAXREUSE	integer	integer other than zero	maximum number of re-used sensitivities per iteration
SENALLCALCINT	integer	greater than one	iteration interval at which all sensitivities re-calculated
SENPREDWEIGHT	real	any number	weight to assign to prediction in computation of composite parameter sensitivities to determine sensitivity re-use
SENPIEXCLUDE	test	"yes" or "no"	include or exclude prior information when computing composite parameter sensitivities to determine sensitivity re-use

Variables required for each parameter group in "parameter groups" section of PEST control file.

Variable	Type	Values	Description
PARGPNME	text	12 characters or less	parameter group name
INCTYP	text	"relative", "absolute", "rel_to_max"	method by which parameter increments are calculated
DERINC	real	greater than zero	absolute or relative parameter increment
DERINCLB	real	zero or greater	absolute lower bound of relative parameter increment
FORCEN	text	"switch", "always_2", "always_3", "switch_5", "always_5"	determines whether central derivatives calculation is undertaken, and whether three points or four points are employed in central derivatives calculation
DERINCMUL	real	greater than zero	derivative increment multiplier when undertaking central derivatives calculation
DERMTHD	text	"parabolic", "outside_pts", "best_fit", "minvar", "maxprec"	method of central derivatives calculation
SPLITTHRESH	real	greater than zero (or zero to deactive)	slope threshold for split slope analysis
SPLITRELDIFF	real	greater than zero	relative slope difference threshold for action
SPLITACTION	text	text	"smaller", "zero" or "previous"

Variables required for each parameter in "parameter data" section of PEST control file.

Variable	Type	Values	Description
PARNME	text	12 characters or less	parameter name
PARTRANS	text	"log", "none", "fixed", "tied"	parameter transformation
PARCHGLIM	text	"relative", "factor", or absolute(N)	type of parameter change limit
PARVAL1	real	any real number	initial parameter value
PARLBND	real	less than or equal to PARVAL1	parameter lower bound
PARUBND	real	greater than or equal to PARVAL1	parameter upper bound
PARGP	text	12 characters or less	parameter group name
SCALE	real	any number other than zero	multiplication factor for parameter
OFFSET	real	any number	number to add to parameter
DERCOM	integer	zero or greater	model command line used in computing parameter increments
PARTIED	text	12 characters or less	the name of the parameter to which another parameter is tied

Variables required for each observation group in "observation groups" section of PEST control file.

Variable	Type	Values	Description
OBGNME	text	12 characters or less	observation group name
GTARG	real	positive	group-specific target measurement objective function
COVFILE	text	a filename	optional covariance matrix file associated with group

Variables required for each observation in "observation data" section of PEST control file.

Variable	Type	Values	Description
OBSNME	text	20 characters or less	observation name
OBSVAL	real	any number	measured value of observation
WEIGHT	real	zero or greater	observation weight
OBGNME	text	12 characters or less	observation group to which observation assigned

Variables in optional "derivatives command line" section of PEST control file.

Variable	Type	Values	Description
DERCOMLINE	text	system command	command to run model for derivatives calculation
EXTDERFLE	text	a filename	name of external derivatives file

Variables in "model command line" section of PEST control file.

Variable	Type	Values	Description
COMLINE	text	system command	command to run model

Variables in "model input/output" section of PEST control file.

Variable	Type	Values	Description
TEMPFLE	text	a filename	template file
INFLE	text	a filename	model input file
INSFLE	text	a filename	instruction file
OUTFLE	text	a filename	model output file

Variables in "prior information" section of PEST control file.

Variable	Type	Values	Description
PILBL	text	20 characters or less	name of prior information equation
PIFAC	text	real number other than zero	parameter value factor
PARNME	text	12 characters or less	parameter name
PIVAL	real	any number	"observed value" of prior information
WEIGHT	real	zero or greater	prior information weight
OBGNME	text	12 characters or less	observation group name

Variables in optional "predictive analysis" section of PEST control file.

Variable	Type	Values	Description
NPREDMAXMIN	integer	-1 or 1	maximise or minimise prediction
PREDNOISE	integer	0 or 1	instructs PEST to include predictive noise in prediction
PD0	real	greater than zero	target objective function
PD1	real	greater than PD0	acceptable objective function
PD2	real	greater than PD1	objective function at which Marquardt lambda testing procedure is altered as prediction is maximised/minimised
ABSPREDLAM	real	zero or greater	absolute prediction change to terminate Marquardt lambda testing
RELPREDLAM	real	zero or greater	relative prediction change to terminate Marquardt lambda testing
INITSCHFAC	real	greater than zero	initial line search factor
MULSCHFAC	real	greater than one	factor by which line search factors are increased along line
NSEARCH	integer	greater than zero	maximum number of model runs in line search
ABSPREDSWH	real	zero or greater	absolute prediction change at which to use central derivatives calculation
RELPREDSWH	real	zero or greater	relative prediction change at which to use central derivatives calculation
NPREDNORED	integer	one or greater	iterations since prediction raised/lowered at which termination is triggered
ABSPREDSTP	real	zero or greater	absolute prediction change at which to trigger termination
RELPREDSTP	real	zero or greater	relative prediction change at which to trigger termination
NPREDSTP	integer	two or greater	number of iterations over which ABSPREDSTP and RELPREDSTP apply

Variables in optional "regularisation" section of PEST control file.

Variable	Type	Values	Description
PHIMLIM	real	greater than zero	target measurement objective function
PHIMACCEPT	real	greater than PHIMLIM	acceptable measurement objective function
FRACPHIM	real	optional; zero or greater, but less than one	set target measurement objective function at this fraction of current measurement objective function
MEMSAVE	text	"memsave" or "nomemsave"	activate conservation of memory at cost of execution speed and quantity of model output
WFINIT	real	greater than zero	initial regularisation weight factor
WFMIN	real	greater than zero	minimum regularisation weight factor
WFMAX	real	greater than WFMAX	maximum regularisation weight factor
LINREG	text	"linreg" or "nonlinreg"	informs PEST that all regularisation constraints are linear
REGCONTINUE	text	"continue" or "nocontinue"	instructs PEST to continue minimising regularisation objective function even if measurement objective function less than PHIMLIM
WFFAC	real	greater than one	regularisation weight factor adjustment factor
WFTOL	real	greater than zero	convergence criterion for regularisation weight factor
IREGADJ	integer	0, 1, 2, 3, 4 or 5	instructs PEST to perform inter-regularisation group weight factor adjustment, or to compute new relative weights for regularisation observations and prior information equations
NOPTREGADJ	integer	1 or greater	the optimisation iteration interval for re-calculation of regularisation weights if IREGADJ is 4 or 5
REGWEIGHTRAT	real	absolute value of 1 or greater	the ratio of highest to lowest regularisation weight; spread is logarithmic with null space projection if set negative
REGSINGTHRESH	real	less than 1 and greater than zero	singular value of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ (as factor of highest singular value) at which use of higher regularisation weights commences if IREGADJ is set to 5

Variables in optional "pareto" section of PEST control file.

Variable	Type	Values	Description
PARETO_OBSGROUP	text	12 characters or less	name of observation group whose weights are subject to multiplication by a variable weight factor
PARETO_WTFAC_START	real	zero or greater	initial weight factor for user-specified observation group
PARETO_WTFAC_FIN	real	greater than PARETO_WTFAC_START	final weight factor for user-specified observation group
NUM_WTFAC_INT	integer	greater than zero	number of weight factor increments to employ in traversing Pareto front
NUM_ITER_START	integer	zero or greater	number of optimisation iterations to employ when using intial weight factor
NUM_ITER_GEN	integer	greater than zero	number of optimization iterations to employ when using any weight factor other than PARETO_WTFAC_START or PARETO_WTFAC_FIN
NUM_ITER_FIN	integer	zero or greater	number of optimization iterations to employ when using final weight factor
ALT_TERM	integer	zero or one	set to one in order to activate PEST termination determined by value of a specified model output
OBS_TERM	text	20 characters or less	the name of an observation cited in the "observation data" section of the PEST control file whose value will be monitored for possible PEST run termination
ABOVE_OR_BELOW	text	"above" or "below"	determines whether the monitored model output must be above or below the threshold to precipitate run termination
OBS_THRESH	real	any number	value that monitored model output must exceed or undercut to precipitate model run termination
ITER_THRESH	integer	zero or greater	the number of optimization iterations for which the model output threshold must be exceeded or undercut to precipitate run termination
NOBS_REPORT	integer	0 or greater	number of model outputs whose values to report
OBS_REPORT_N	text	20 characters or less	the name of the N'th observation whose value is reported in the POD and PPD files written by PEST when run in "pareto" mode

Table of Contents

1. Introduction

This document describes alterations to PEST that have been made since publication of the fifth edition of the manual. Publication of this edition of the manual coincided with the release of version 9.0 of PEST. Hence all alterations documented herein pertain to versions later than that.

Of particular importance are additions to PEST functionality, and additions to the PEST utility suite, that implement improved regularised inversion and model predictive error variance analysis.

The author would like to point out that he is aware of the fact that when an addendum to a manual grows to a size that is nearly as large as the manual itself, it is time to write a new edition to the manual. This will take place in the near future. In fact, there will be two manuals – a "beginners manual" and an "advanced manual". But for now this has to wait. There are many things I would like to do, and this is surely one of them. However in the meantime the user can perhaps remind him/herself that this inconvenience is part of the price that must be paid for the fact that PEST is in the public domain.

While, for the author, the making of improvements to PEST and its utility suite is largely a labour of love, it must be pointed out that I have had financial help for certain, important, aspects of this work. I would particularly like to acknowledge the following.

- Work related to inclusion of the LSQR solver in PEST, and to calibration-constrained Monte Carlo analysis, was supported by a contract with Boise State University under EPA Grant X-96004601-0. This contract also supported the development of software to implement an interface between PEST and the adjoint process of MODFLOW-2005 developed by Tom Clemo of Boise State University (see the manual to the PEST Groundwater Data Utilities for more details).
- Support from S. S. Papadopulous and Associates is gratefully acknowledged.
- So too, is support from EMS-I.
- The INFSTAT utility documented herein was written under contract to the Madison, WI, office of USGS.
- Help in development of the JACTEST utility and of the CMAES_P global optimizer was provided by Los Alamos National Laboratory and South Florida Water Management District.

2. Alterations and Upgrades to PEST

2.1 New Restart Option

Execution of PEST can now be recommenced with four different switches. Use of the "/r", "/j" and "/s" switches is documented in the PEST manual. The new restart switch is the "/d" switch.

The "/d" switch has a similar role to the existing "/s" switch. Recall that the "/s" switch can be used to recommence Parallel PEST execution at the same model run in which it was previously interrupted, if interruption took place during calculation of the Jacobian matrix. If Parallel PEST execution was interrupted during the Marquardt lambda testing procedure however, restarting PEST with the "/s" switch has the same effect as restarting it with the "/j" switch; that is, PEST is restarted at that point in its previous run at which calculation of the Jacobian matrix was just completed.

If PEST is run in non-parallel mode, then it cannot be restarted with the "/s" switch. However it can be re-started with the "/d" switch. The effect is exactly the same as re-starting a Parallel PEST run with the "/s" switch.

Note the following important points pertaining to PEST's restart switches:-

1. Parallel PEST cannot be restarted with the "/d" switch.
2. Non-Parallel PEST cannot be restarted with the "/s" switch.
3. A previously stopped Parallel PEST run can be restarted as non-Parallel PEST or Parallel PEST using either the "/r" or "/j" switches.
4. A previously stopped non-Parallel PEST run can be restarted as Parallel PEST using either the "/r" or "/j" switches.
5. A previously stopped Parallel PEST run cannot be restarted as non-Parallel PEST with the "/s" switch.
6. A previously stopped non-Parallel PEST run cannot be restarted as Parallel PEST with the "/d" switch.

The reader is probably wondering at this stage why, if the "/s" and "/d" switches have the same effect, they cannot be used interchangeably between Parallel and non-Parallel PEST. The reason is that run management is very different between these two versions of PEST. For Parallel PEST, runs are not necessarily undertaken in sequence. Run results in the restart file are also not stored in sequence. For non-Parallel PEST, the restart file does not hold run results; rather it holds fragments of the Jacobian matrix. This can save considerably on file storage when central differences are used for derivatives calculation. Also, this restart methodology is more easily combined with external derivatives calculation – this being available only in the non-Parallel version of PEST at present.

It is freely admitted however, that an all-purpose run manager is needed. This, hopefully, will be one an outcome of future PEST development. At that stage the "/s" and "/d" switches will be combined.

2.2 Template and Instruction File Headers

The first line of a template file must be:-

```
ptf #
```

where "#" can be replaced by any other character that is suitable for definition of a parameter delimiter. "ptf" stands for "PEST template file".

The first line of an instruction file must be:-

```
pif #
```

where "#" can be replaced by any other character that is suitable for use as a marker delimiter. "pif" stands for "PEST instruction file".

PEST and its checking utilities now allow alternative headers for these two files. "jtf" (for "JUPITER template file") can be used in place of "ptf", and "jif" (for "JUPITER instruction file") can be used in place of "pif".

2.3 User Intervention

The "parameter hold file" can be used to hold the values of certain parameters fixed. By stopping and restarting PEST using the "/j" switch, an attempt can then be made to re-calculate the parameter upgrade vector with troublesome parameters temporarily deactivated.

As is explained in the PEST manual, the parameter hold file can be used not just for holding parameters fixed, but also for altering the values of certain control variables. At the time of writing, control variables for which in-flight alterations are allowed are LAMBDA, RELPARMAX, FACPARMAX and UPVECBEND.

Five new control variables can now be altered through the parameter hold file. These all pertain to the predictive analysis process. They are INITSCHFAC, MULSCHFAC, NSEARCH, RELPREDSTP and ABSPREDSTP.

The first three of these variables govern operation of the line search procedure used in prediction maximisation/minimisation. On many occasions, use of this procedure is critical to the success of the maximisation/minimisation process. Contrary to advice initially supplied with PEST documentation, it is often best to start this procedure with a low INITSCHFAC value (for example 0.2). However if it is set too low, this can result in wasted model runs. Similarly, MULSCHFAC may need to be set lower than 2.0 for highly nonlinear cases (even as low as 1.3). As this may require increased runs per line search, NSEARCH may need to be set higher than normal.

It is not explained in the PEST manual that termination criteria for the line search are the same as those for termination of the predictive analysis process as defined by the RELPREDSTP and ABSPREDSTP variables. (Actually, the line search procedure follows a complex algorithm in which both the objective function and prediction value are monitored, with the former often expected to fall and then rise along the path of the search; hence there are, in fact, a number of other termination criteria required for implementation of this algorithm). It has been found from experience that in difficult problems, RELPREDSTP and ABSPREDSTP may need to be set quite low. This will be particularly important where small predictive uncertainty exists in a large predictive number (in which case the absolute, rather than the relative, termination criterion should be used), especially where line search increments are chosen to be small for better accommodation of nonlinear models.

New values for the above predictive analysis control variables can be supplied through the parameter hold file in the same way as for other variables. The following figure shows a file in which new values are provided for all of them.

As discussed in the PEST manual, the parameter hold file must be named *case.hld* where *case* is the filename base of the pertinent PEST control file. The hold file is read just after calculation of the Jacobian matrix and just before calculation of parameter upgrades. It is very important that a parameter hold file be deleted after use; this will prevent its contents being inadvertently used on a subsequent PEST run.

```
initschfac 0.2
mulschfac 1.5
relpredstp 1.0e-4
abspredstp 0.00
```

Example of a parameter hold file.

2.4 Improvements to the Predictive Analyser's Line Search

Considerable improvements have been made to the line search implemented by PEST's predictive analyser. Recall from the PEST manual that implementation of a line search to find the highest/lowest prediction for which the objective function is at or lower than the user-supplied limit is optional; it is implemented only if NSEARCH is set to a number greater than 1. It has been found, however, that use of this option can greatly increase PEST's ability to find the maximum/minimum constrained predictive value, especially in difficult numerical circumstances such as those that prevail where parameters are large in number and are highly correlated. The line search algorithm implemented by PEST is actually quite complicated, for not only does PEST need to monitor the value of the prediction as it travels along a certain parameter trajectory; it must also monitor the value of the objective function as well. On many occasions the objective function may actually fall before it rises to PD0 (the user-defined objective function limit) when travelling along the parameter upgrade line from the current set of parameter values to a potential new set of values.

It has been found that in those situations where numerical complexity dictates that a line search is warranted, it is worth doing this line search thoroughly. To optimise the efficacy of this search, the following settings are suggested.

- Set INITSCHFAC (initial line search factor) to 0.2 or 0.3; thus the line search begins at a point on the potential parameter upgrade line which is not too far from current parameter values.
- Set MULSCHFAC (line search factor multiplier) between 1.3 and 1.7.
- Set NSEARCH (maximum number of model runs devoted to the line search) to 15.
- Set PD1 quite close to PD0, maybe only 0.2% or less higher.
- Set ABSPREDSTP and RELPREDSTP reasonably tight. (These are actually the termination criteria for the predictive analysis process; one tenth of these values is also used for termination of the line search.)
- Start the predictive analysis process from optimised parameter values (for which the objective function is less than PD0).

Because the line search is repeated for every different value of the Marquardt lambda tested, it can consume an inordinate number of model runs if significant Marquardt lambda adjustment is warranted. Code has been inserted within PEST that aims to reduce the number of line search runs required when testing Marquardt lambdas after the first on any one optimisation iteration. Nevertheless, any savings that can be made in reducing trial Marquardt lambdas will result in increased efficiency. Thus after you have had experience with using the predictive analyser on your particular task, you may wish to consider setting the initial Marquardt lambda (RLAMBDA1) lower or higher than you normally would, if this is where PEST seems to prefer its value to be. Alternatively, set NUMLAM to 1, so that only 1 Marquardt lambda is employed per optimisation iteration; if this strategy is adopted it is probably good practice to try a very low value for RLAMBDA1, maybe in the vicinity of 0.01 to 0.001 (or even zero), and to start the predictive analysis process from previously-optimised parameter values.

While conducting a line search is a very time-consuming activity, experience has shown that it can be worth the effort in many circumstances; it is sometimes quite surprising how high or low calibration-constrained predictions can be. The cost of finding these extreme predictions, however, can be particularly high when using Parallel PEST to undertake the predictive analysis process, because while Jacobian runs are parallelised, line search runs are not. It is planned, however to partially parallelise this process in the future.

2.5 Switch to Three-Point Derivatives Calculation

A new variable has been introduced to the "control data" section of the PEST control file in order to provide more flexibility to the way in which PEST chooses (or not) to switch to central derivatives calculation.

The eighth line of the PEST control file must supply a value for the PHIREDSWH variable; this is the first (and optionally the only) variable on this line. If FORCEN for any parameter group is set to "switch", PEST will switch to central derivatives calculation on the first occasion on which the relative objective function improvement is less than PHIREDSWH. Thus, for example, if PHIREDSWH is set to 0.1 (which is its suggested value), and if, at the end of any particular optimisation iteration, the new objective function is greater than 90% of the objective function at the beginning of the iteration, PEST will employ three-point derivatives calculation for the remainder of the optimisation process. As a result of this, up to twice as many model runs per iteration will be required for filling of the Jacobian matrix. With more accurately calculated derivatives, PEST will often fair better in lowering the objective function further, especially in contexts where parameter insensitivity or correlation creates an ill-conditioned normal matrix.

There are times, however, where PEST "trips" into central derivatives calculation before increased derivatives accuracy is really needed. This can occur, for example, where some parameters need to change a great deal before they can effect a noticeable lowering of the objective function, but they are prevented from doing so (or prevent other parameters from doing so) because of the action of the parameter upgrade limiting variables RELPARMAX and FACPARMAX. In cases like this, model run efficiency would be better served if the parameter estimation process were continued with forward difference derivatives calculation until the offending parameter(s) have moved a sufficient distance in parameter space for their effect on the objective function to be noticeable, or for PEST not to have to limit their movement (and with it the movement of other parameters) in order to curtail excessive parameter variations within the one iteration (an often necessary measure for the prevention

of instability in highly nonlinear cases). Instead, the premature introduction of central derivatives calculation simply increases the number of model runs required for completion of the parameter estimation process, with no real benefits to this process being incurred from three-point derivatives calculation from having been introduced so early.

A new variable named NOPTSWITCH may now optionally follow PHIREDSWH on the eighth line of the PEST control file. If supplied, this must be an integer equal to 1 or greater. If it is greater than 1, PEST will not switch to central derivatives calculation until the NOPTSWITCH'th iteration at least, as long as the objective function does not rise during any optimisation iteration. If the objective function does, in fact, rise, then the NOPTSWITCH setting is overridden and PEST switches to three-point derivatives calculation.

If the optional DOAUI variable is supplied on the eighth line of the PEST control file it must follow NOPTSWITCH, if NOPTSWITCH is also supplied. If NOPTSWITCH is not supplied, it must simply follow PHIREDSWH.

2.6 SVD-Assist: Super Parameter Definition

Recall from the PEST manual that the SVDAPREP utility writes a PEST control file in which super parameters are defined which can then be used for SVD-assisted parameter estimation. In writing this file, SVDAPREP transfers observations and observation weights directly from the base parameter PEST control file to the new super parameter PEST control file. When PEST then commences an SVD-assisted parameter estimation run, it undertakes singular value decomposition of the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix in order to define the linear combination of base parameters which comprises each super parameter. Singular value decomposition and super parameter definition is only then repeated if a base parameter hits its bound; that base parameter then remains at its bound while the super parameter estimation process proceeds in order to estimate other base parameters.

In versions of PEST prior to 9.2, singular value decomposition for the purpose of super parameter definition took place on the basis of base parameters defined in the base parameter PEST control file, and observations and weights defined in the super parameter PEST control file. Under normal circumstances, observations and weights defined in the super parameter control file are the same as those defined in the base parameter control file. However for versions of PEST from 9.2 onwards, the user has the option of altering observations and weights defined in the super parameter control file once this file has been built by SVDAPREP. Furthermore, super parameter definition now takes place on the basis of observations and weights contained in the original base parameter control file. Hence parameter estimation can take place on the basis of a new set of observations and/or weights, different from those used for definition of super parameters.

There may be some situations where the ability to undertake super parameter definition on the basis of one set of observations and weights, and parameter estimation on the basis of another is important. For example, a simple but important instance where this may prove useful is where a covariance matrix is supplied for super parameter estimation based on the observation correlation structure induced by using super parameters. As discussed by Cooley (2004) and by Moore and Doherty (2006) all forms of regularisation, or parameter averaging, lead to a correlated "measurement noise" structure. If it is desired that account be taken of this noise structure in the parameter estimation process, SVDAPREP can be used for generation of the super parameter PEST control file in the usual fashion. Then an appropriate covariance matrix can be supplied for the observations contained in that file. However if individual weights, rather than a covariance matrix, are used in the base PEST control file,

super parameter definition takes place on the basis of the individual weights, while estimation of super parameters (and hence indirectly of base parameters) takes place using a covariance matrix which best characterizes "measurement noise" in these circumstances. In more complex modelling contexts, the number, names and types of observations can be different in the super parameter PEST control file from those contained in the base parameter PEST control file.

The following points must be born carefully in mind when using PEST for SVD-assisted parameter estimation.

1. When PEST commences a super parameter estimation run by undertaking singular value decomposition on the basis of observations and weights contained in the base PEST control file, it includes all observations and prior information equations defined in the base PEST control file in formulation of the **X** matrix if PESTMODE is set to "estimation" in that file. However if, in the base PEST control file, PESTMODE is set to "regularisation", then observations and prior information equations belonging to regularisation groups are excluded from this matrix. If PESTMODE is set to "prediction" in the base PEST control file, the sole member of the observation group "predict" is also excluded from the **X** matrix.
2. If prior information is used in the base PEST control file, but prior information sensitivities are not available from the corresponding Jacobian matrix file, PEST will simply not use sensitivities pertaining to that prior information in the formulation or super parameters. (This can occur when initial base parameter sensitivities were calculated using SVD without prior information in an attempt to evaluate a suitable number of base parameters to employ – see the PEST manual for details.)
3. While covariance matrices can be supplied for any or all observation and prior information groups in the super parameter PEST control file, a covariance matrix can only be supplied for observation and prior information groups whose names begin with "regul" in the base PEST control file, and then only if PESTMODE is set to "regularisation".

2.7 SVD-Assist: Multiple Parameter Value Files

The SVD-assist section of the PEST control file now has an optional third line; this line contains control variables which can govern the operation of certain aspects of the SVD-assisted parameter estimation process. The name of the first variable on this line is SVDA_MULBPA. The figure below shows an example of the SVD-assist section of a PEST control file with the extra new line containing only this variable.

```
* svd assist
temp1.pst
temp1.jco
1
```

The SVD-Assist section of a PEST control file.

SVDA_MULBPA can be set to either 0 or 1. If it is set to 1, then not just one, but a series of "BPA files" will be recorded during the course of the parameter estimation process. Recall that a "BPA file" contains best base parameter estimates at any stage of the parameter estimation process; at the end of the parameter estimation process it contains best base parameters achieved throughout the entire process. Recall also that the filename base of the BPA file is that of the base parameter PEST control file, not the super parameter control file

through which parameter estimation actually takes place.

With SVDA_MULBPA set to 1, PEST records two BPA files on every occasion that base parameter values are improved. The first of these files is the normal BPA file. The second has the same name, but with an extension of ".N" appended, with "N" indicating the optimisation iteration number in which the base parameters recorded therein were achieved. Thus if, for example, the filename base of a base PEST control file is ***base***, then at the conclusion of the parameter estimation process BPA files named *base.bpa, base.bpa.1, base.bpa.2* etc will be present within the current working directory. The first of these contains the overall best base parameters achieved through the parameter estimation process. The others contain base parameter values achieved during the indicated optimisation iterations. Note that not all optimisation iterations will be represented in this sequence – only those iterations will be represented where base parameters are improved from those previously achieved during the current parameter estimation process.

To accommodate this new option, the SVDAPREP utility has one extra prompt. viz:

```
Write multiple BPA files?  [y/n] (<Enter> if "n"):
```

As is apparent, the default is "no" (and SVDA_MULBPA is thus set to zero), leaving PEST operation unchanged from previous versions.

2.8 SVD-Assist: Improved Handling of Untransformed Parameters

2.8.1 General

Problems can arise when log-transformed and untransformed parameters are simultaneously estimated using SVD-assist. They are an outcome of the fact that log-transformed parameters can have vastly different sensitivities from those of untransformed parameters. In fact the latter can have vastly different sensitivities from those of other untransformed parameters. PEST has internal numerical means of accommodating the estimation of parameters of vastly differing sensitivity. However special considerations arise when implementing SVD-assisted parameter estimation. This section describes those considerations, and the functionality that is now available within PEST to accommodate them.

Ideally, when formulating an SVD-assisted parameter estimation problem, parameters should be normalized by their standard deviations. There are a number of reasons for this, one of which is that the solution to the regularized inverse problem can then approach the solution of maximum parameter likelihood (if initial parameter values are chosen wisely). Another important consideration is that normalisation by standard deviation tends to equalize sensitivities across different parameter types. If, after such normalisation, a particular base parameter, or group of base parameters, is found to be still insensitive, such parameters will be removed from consideration in the normal course of formulating the SVD-assisted parameter estimation process on the basis of eigenvectors corresponding to the major singular values of $\mathbf{X}^t\mathbf{QX}$; thus such parameters will not feature strongly in super parameter definition. On the other hand, base parameters which are particularly sensitive after normalisation will feature strongly in super parameter definition, and hence will be adjusted during the parameter estimation process in preference to other, less sensitive, base parameters.

If untransformed base parameters are not normalised, it is possible that some of them may be "artificially hypersensitive". For example in a groundwater model, recharge (which is best

left untransformed to increase the linearity of the parameter estimation process) is often far more sensitive than other parameters (often by many orders of magnitude) simply because low numbers are used to represent it in many modelling contexts. Thus recharge base parameters feature strongly in super parameters. Furthermore in many cases the top few resulting super parameters can be hundreds of times more sensitive than the next super parameters, this creating numerical problems for PEST as it tries to adjust both the recharge-dominated super parameters, together with super parameters of far lower sensitivity which contain linear combinations of other base parameters which may be for more instrumental in lowering the objective function than recharge parameters. If recharge parameters were normalized with respect to their natural variability this problem would not occur, for a very low number expressing what may be a comparatively high recharge value would then become a comparatively high number. Because variations in this high number then have smaller effects on model outputs than variations in the corresponding unnormalised very small number, the extreme sensitivities associated with the latter are removed. A similar affect would be achieved if recharge were log transformed. Mathematically, derivatives of log-transformed parameters are automatically normalized by their values – this achieving something approaching normalization by natural variability. However, in the case of recharge, while log transformation may remove artificial hypersensitivity, the parameter estimation process may suffer because of the nonlinearities introduced to it through log transformation.

Another problem that arises with "artificial hypersensitivity" of base parameters is that it is far too easy for them to encounter their bounds in the course of the parameter estimation process. Recall that it is an unfortunate necessity of employing SVD-assist as an inversion mechanism that once a base parameter hits its bound, it must stay stuck there forever. Premature bounds collisions can be mitigated by employing a relatively low RELPARMAX setting. However even this measure cannot forestall wholesale bounds collision of hypersensitive parameters, often on the first iteration of the parameter estimation process. The reason for this is that super parameters are actually the coefficients by which normalised (i.e. with magnitude 1) eigenvectors of $\mathbf{X}^t\mathbf{QX}$ are multiplied in determining parameter combinations which best reduce the (measurement) objective function. The component of a hypersensitive base parameter in one of these normalised eigenvectors can be such that, even when the corresponding super parameter is increased only incrementally for the purpose of derivatives calculation, the pertinent base parameter can hit its bounds; this happens solely as a consequence of the fact that the eigenvector has a magnitude of 1 and that hypersensitive base parameters (which will dominate the top few eigenvectors) have a "natural magnitude" which may be orders of magnitude smaller than this. Obviously this is an outrageous situation which must be prevented at all costs. Fortunately, the problem is easily forestalled by estimating high multiples of hypersensitive base parameters - that is, by appropriate base parameter scaling.

Base parameter scaling can be easily undertaken by direct editing of a base PEST control file prior to running SVDAPREP. Say, for example, that all parameters of a certain type (for example all recharge parameters) are to be scaled so that they are a thousand times less sensitive, thus avoiding the deleterious affects of artificial base parameter hypersensitivity discussed above. This could be achieved in the following manner.

1. Provide all recharge parameters with a SCALE of 0.001 in the "parameter data" section of the base PEST control file.
2. Multiply all recharge parameter initial values, and their bounds, by a factor of 1000.

3. Multiply any absolute derivative increments, and relative increment absolute lower bounds, by a factor of 1000 in the "parameter groups" section of the PEST control file.
4. Alter any prior information equations containing these parameters such that they are correct when any recharge parameter involved in any equation is multiplied by 1000.

Thus PEST "sees" (and estimates) 1000 times each recharge parameter, while the model "sees" correct recharge parameter values. (Note that if an OFFSET is employed for a parameter this does not need to be altered because PEST multiplies by the SCALE before it adds the OFFSET. Note also that if these changes are made to a base parameter control file for which a corresponding JCO file already exists, JCO2JCO can be used to create a new JCO file corresponding to the edited PEST control file, thus obviating the need for re-calculation of base parameter sensitivities). As a result of these alterations to the base PEST control file, sensitivities with respect to these now-scaled recharge base parameters will be 1000 times less than those of their unscaled counterparts.

The above procedure is recommended if programs such as RESPROC and its partner utilities are to be used for post-calibration predictive error variance analysis, for the user can craft the parameter estimation problem to suit his/her purposes, and have full control over the scaling process. However PEST allows somewhat equivalent operations to be performed with less trouble than the above as a practical means of ensuring good performance of the SVD-assisted parameter estimation process. These will now be described.

2.8.2 The SVDA_SCALADJ Control Variable

PEST now provides a variety of automatic base parameter scaling mechanisms to combat the problems associated with base parameter hypersensitivity outlined above. The variable which controls selection of these mechanisms is named SVDA_SCALADJ. If present, it is situated after the SVDA_MULBPA variable on the third line of the "svd assist" section of the super parameter PEST control file. See the figure below in which a value of "4" is assigned to this variable. Permissible values are -4, -3, -2, -1, 0, 1, 2, 3, and 4. If omitted from the PEST control file, a default value of 0 is assumed; in this case no base parameter scaling is undertaken.

```
* svd assist
temp1.pst
temp1.jco
1 4
```

The SVD-assist section of a super parameter PEST control file in which the SVDA_SCALADJ control variable is assigned a value of 4.

Base parameter scaling options are now discussed in detail. It is important to note that all scaling is undertaken internally by PEST. Estimated parameter values as displayed, for example, in the BPA file do not require "de-scaling" by the user.

2.8.2.1 Non-Log-Transformed Parameter Scale Adjustment by Group

As stated above, problems caused by base parameter hypersensitivity do not tend to occur when implementing SVD-assisted parameter estimation where all base parameters are log-transformed. This is a result of the "self-normalizing" effect of logarithmic transformation. As an inspection of the SEN (composite sensitivity) file produced by PEST often reveals, sensitivities of log-transformed parameters, even of vastly different types, are generally not as variable as those of untransformed parameters.

One option for untransformed parameters is to scale them such that their sensitivities are roughly equal to those of log-transformed parameters involved in the parameter estimation process. If SVDA_SCALADJ is set to 1 PEST undertakes scaling of untransformed base parameters in the following fashion.

1. First it calculates the composite sensitivity of every adjustable base parameter; this is calculated as the magnitude of the column of the Jacobian matrix corresponding to each parameter, with each entry in that column multiplied by the squared weight associated with the corresponding observation.
2. Next it calculates the average composite sensitivity of all parameters in each parameter group; that is, it calculates a kind of parameter group composite sensitivity. It is assumed that all adjustable parameters within any parameter group are either log-transformed or untransformed (tied and fixed members of a particular parameter group are permitted).
3. For each untransformed parameter group PEST evaluates a scaling factor, applied to all parameters in the group, which equates the average composite sensitivity for that group to that of the log-transformed group of maximum average composite sensitivity.

Variations of this theme are as follows.

1. If SVDA_SCALADJ is set to -1, PEST will never *increase* the sensitivity of an untransformed parameter group to ensure composite sensitivity equalisation with the log-transformed parameter group of highest average composite sensitivity; it will only *decrease* the sensitivity of any untransformed parameter group.
2. If SVDA_SCALADJ is set to 2, PEST calculates the average composite sensitivity of *all* log-transformed parameters, irrespective of the parameter group to which they belong. Then, for each untransformed parameter group, all member parameters are scaled such that the average composite sensitivity of the parameter group is equal to that of all log-transformed parameters.
3. If SVDA_SCALADJ is set to -2, PEST will never *increase* the sensitivity of an untransformed parameter group to ensure composite sensitivity equalisation with log-transformed parameters; it will only *decrease* the sensitivity of an untransformed parameter group.

PEST will terminate execution with an error message if SVDA_SCALADJ is set to -2, -1, 1 or 2 and any of the following conditions are met.

1. A base parameter group exists which contains both log-transformed and untransformed parameters.
2. No log-transformed parameters are cited within the base control file.
3. All log-transformed base parameters have zero sensitivity.

2.8.2.2 Individual Non-Log-Transformed Base Parameter Scale Adjustment

As stated above, ideally parameters should be scaled by their standard deviations - that is, by their innate variability – that is, by the uncertainty associated with those parameters in the absence of any calibration information, this uncertainty being based solely on specialist knowledge of the system. Such information cannot be supplied directly on a PEST control file. However parameter upper and lower bounds (which *are* supplied through a PEST control

file) can be considered as surrogates for information of this type.

If SVDA_SCALADJ is set to 3, PEST assumes that the upper and lower bounds of all untransformed parameters are set approximately four standard deviations apart. Untransformed parameters are then scaled by standard deviation calculated accordingly. (There is no need for exactness here as the success of scaling in improving the performance of SVD-assisted parameter estimation is not unduly sensitive to the exact scaling values. Four standard deviations is a convenient assumption as, for normally distributed parameters, it corresponds to a confidence interval of about 95 percent.)

If SVDA_SCALADJ is set to -3, PEST will not apply a scale factor that *increases* the sensitivity of a parameter; it will only apply a scale factor that *reduces* the sensitivity of an individual untransformed base parameter.

2.8.2.3 Pervasive Base Parameter Scale Adjustment

If SVDA_SCALADJ is set to 4, the strategy implemented when SVDA_SCALADJ is set to 3 is extended to accommodate all adjustable base parameters – even those that are log-transformed. As for untransformed parameters, it is assumed that the bounds on log-transformed parameters represent a separation of about 4 standard deviations, and that their probability distributions are log-normal. Once again, these assumptions are not critical by any means; they simply represent a convenient basis for scaling. If SVDA_SCALADJ is set to -4, no base parameter sensitivities will be increased in the process of base parameter scale adjustment.

2.8.2.4 Which Option to Use

Option 4 is the preferred option, both theoretically and in terms of optimisation of PEST performance when carrying out SVD-assisted parameter estimation. However it requires that the user pay more careful attention to the setting of parameter bounds than he/she may normally do. In fact on many occasion of PEST usage, scant attention is given to parameter bounds at all, these being set wide enough apart for PEST to "listen to what the data says" about the system being simulated by the model, and about the model itself, through the values that it assigns to parameters employed by the model, unencumbered by the necessity to respect bounds placed on these parameters. This, indeed, can be an important role played by the calibration process, for if PEST insists on assigning unacceptable values to certain model parameters, this may indicate conceptual flaws in the model. If all parameters are log-transformed, setting SVDA_SCALADJ to 4 with parameter bounds set wide apart should not cause problems (though it would be better to set it to zero under these conditions). However if some parameters are not log transformed, use of option 4 could lead to dreadful performance of the SVD-assisted parameter estimation process, as hypersensitivity of one or more types of untransformed parameter could result in rapid bounds collision for the reasons outlined above.

Notwithstanding superior performance of a properly-constructed SVD-assisted parameter estimation process with SVDA_SCALADJ set to 4, the SVDAPREP utility (which prepares a super parameter PEST control file from a base parameter PEST control file) provides a default setting of 2 for SVDA_SCALADJ. Experience has shown that SVD-assisted parameter estimation when both log-transformed and untransformed parameters are simultaneously estimated is not as robust with this option as it is for options 3 or 4 with careful consideration given to the setting of parameter bounds. However if base parameter bounds are set far apart in order to reduce their effect on the parameter estimation process

(either deliberately for the reasons outlined above, through user-laziness, or even as a result of the unlikely, but not impossible, occurrence that a user did not read this section of the manual), then use of options 3 or 4 may have disastrous consequences for the SVD-assisted parameter estimation process. Hence option 2 is the safest SVDAPREP option. The user should not forget, however, the benefits of setting SVDA_SCALADJ to 4 in conjunction with a careful bounds setting strategy.

2.9 SVD-Assist: Externally-Defined Super Parameters

Total flexibility in definition of super parameters is available through having the eigenvectors with which these super parameters are associated defined without reference to a PEST control file, and its associated Jacobian matrix, altogether. This can be achieved through setting a new SVD-Assist control variable SVDA_EXTSUPER to 1, and through supplying a "super parameter definition matrix" in place of a base Jacobian matrix. The following example shows the SVD-Assist section of a PEST control file in which these settings are made.

```
* svd assist
temp1.pst
super.mat
1 0 1
```

The SVD-assist section of a super parameter PEST control file in which the SVDA_EXTSUPER control variable is set to 1, and the name a super parameter definition file is supplied in place of that of a base Jacobian matrix file.

The SVDA_EXTSUPER control variable is an integer which must be supplied as either 0 or 1. If omitted it is assumed to be zero.

If SVDA_EXTSUPER is set to 1 PEST does not undertake singular value decomposition of an $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix for definition of directions in parameter space with which super parameters are associated. Instead it reads these directions from a matrix file; the format for such a file is described later in this addendum. If *n* super parameters are featured in a PEST control file, then the directions in parameter space associated with these parameters are defined as the first *n* columns of the matrix supplied in the matrix file. If more than *n* columns are featured in this matrix, columns after the *n*'th are ignored.

The matrix file supplied to PEST as the super parameter definition file must meet certain requirements. As already stated, it must have as many columns (or more) as there are super parameters defined in the super parameter PEST control file; the names associated with these columns are ignored. It must also have at least as many rows are there are base parameters defined in the base parameter PEST control file. Furthermore each adjustable parameter defined in the base PEST control file should have a matrix row associated with it (and linked to it through equality of base parameter name and associated row name) in the super parameter definition matrix file. It is not essential that these names be supplied in the same order in both of these files. Nor does it matter if the super parameter matrix file contains extra rows linked, perhaps, to fixed or tied base parameters, or to other parameters altogether. However it is essential that no adjustable base parameter name be unmatched to a row of the super parameter definition matrix file.

Note the following:

1. **It is important for the sake of calculation of derivatives with respect to base parameters that all columns of the external super parameter definition matrix have a magnitude of approximately 1.** Use of the MATSVD utility described in a

later section can ensure this, at the same time as it ensures orthogonality of super parameter vectors. Simply prepare a candidate matrix based, for example, on what are judged to be the most important parameters and/or parameter combinations, and then employ MATSVD to determine a complementary orthogonal, normalised matrix.

2. If SVDA_EXTJAC is set to 1 there is no role for the SVDA_SCALADJ variable. Hence it is set to zero inside of PEST.
3. The SVD-assisted inversion process handles base parameter bounds imposition slightly differently when super parameter directions are assigned externally, from the way in which bounds imposition is handled when base parameter directions are calculated by PEST. In the latter case super parameter directions are re-defined by PEST when salient base parameters stuck to their bounds and are thus effectively removed from the inversion process. In the former case super parameter re-definition cannot take place. As a result, the inversion process may be a little slower under these circumstances.

SVDAPREP has been slightly modified in order to provide the user with the option of using an external super parameter definition file in place of a base parameter Jacobian file for super parameter definition.

2.10 A New IREGADJ Option

As described in the PEST manual, a value for the IREGADJ variable can optionally be added to the end of the "regularisation" section of the PEST control file; if this variable is not present, its value is assumed to be zero. Experience has demonstrated that setting this variable to "1" can be a very useful means of accommodating the presence of more than one regularisation group within the PEST control file, for it is often a difficult matter for the user to determine the appropriate weightings to use between these different groups. When IREGADJ is set to 1, PEST takes account of both the number and sensitivities of regularisation observations and prior information equations in each group in determining relative inter-regularisation group weighting, so that the contribution made by each group to the overall set of regularisation constraints is "balanced". However its mechanism for calculating these relative weights is by no means foolproof; nor is it such that it would not benefit from user-assistance in some circumstances. An IREGADJ setting of "3" allows such assistance to take place.

For any IREGADJ setting, PEST respects user-supplied relative observation and prior information weights *within* any regularisation group. However it does not respect weights *between* them, for it determines a weight multiplier specific to each group, independent of user-specified relative inter-group regularisation weights as supplied through the PEST control file. Thus when IREGADJ is set to 1 or 2 there is no need for the user to worry about setting relative regularisation group weighting "properly", because PEST overrides this weighting when calculating its own inter-group regularisation weight factors. Hence if a user does not wish that regularisation weights vary within any regularisation group, there is no reason why all regularisation weights should not be supplied with a value of 1.0 in the PEST control file.

If IREGADJ is set to 3, PEST undertakes the same calculations for the purpose of relative group weight factor calculation that it undertakes when IREGADJ is set to 1. However it then undertakes a "final adjustment" of regularisation weights by multiplying them all by user-supplied regularisation weights. Thus if, for example, a user supplies weights for group

regul1 which are twice those for *regul2*, PEST will multiply all weights within group *regul1* by a factor of 2 relative to those in group *regul1 after* having calculated regularisation weights for these groups using the procedure that is normally employed when IREGADJ is set to 1. (All regularisation weights are then multiplied by the global regularisation weight factor before parameter estimates are net upgraded.)

Setting IREGADJ to 3 has the potential to be very useful where there are many regularisation groups. In such circumstances it is difficult for a user to determine a "balanced" set of inter-regularisation group weight factors him/herself; the result may be poor PEST performance as regularisation constraints fail to compensate for data inadequacy in formulating a well-posed inverse problem. Although PEST may be able to do this quite comfortably itself on many occasions using its normal IREGADJ functionality, it may nevertheless be the user's desire that regularisation constraints for some parameters be enforced more strongly than for others. In these circumstances he/she can supply an initial set of weights that reflect this desire. PEST will then alter its IREGADJ-calculated weights accordingly in the manner described above.

It is apparent from the above discussion that it would be unwise for a user to supply regularisation weights which are markedly different from group to group when IREGADJ is set to 3, for the benefits of IREGADJ adjustment will be lost if internally-calculated relative weights are varied too much from the "balanced" set calculated by PEST. It must be remembered in formulating a weights assignment strategy for constructing the PEST control file that PEST's internal weights adjustment procedure will automatically take into account the population of each regularisation group, and the composite sensitivity of each of the observations or prior information equations comprising each regularisation constraint. When IREGADJ is set to 3, it is the user's task when supplying weights to the PEST control file to provide a basis for relative inter-group weights "fine-tuning" that reflects his/her desirer for certain constraints to be enforced more than others. However if such tuning prevents stable solution of the inverse problem, then the desire for stronger enforcement of one set of constraints over another must be abandoned and uniform regularisation weights supplied, or IREGADJ set to 1.

2.11 Parallel PEST Run Repeats

If Parallel PEST encounters a problem in reading a model output file from a slave's subdirectory, it tries a number of times to read the file before giving up. Then, just in case the problem originated in network congestion or some other troublesome network behaviour, Parallel PEST repeats the model run on the same or another slave. A number of repetitions are attempted before PEST terminates execution with an error message to the screen outlining the nature of the problem encountered.

Where model run times are long and the problem did not in fact originate in network communication failure, this process can take a long time. There are instances where the user would like to be made aware more quickly of such a problem so that he/she can take steps to rectify it if, in fact, the source of the problem is in the model, or one of the components thereof, rather than in the network. Parallel PEST now presents an option through which repeated attempts to run the model are bypassed; instead PEST immediately terminates execution with an appropriate error message. A new variable has been added to the Parallel PEST run management file to activate this option.

The figure below illustrates the construction of a run management file. The second line now contains five variables, the last two of which are optional. If the last variable (RUNREPEAT)

is set to 0, attempts at model run repetition as described above will not take place. If it is set to any other number (or if it is omitted), attempted run repetition will take place.

```
prf
NSLAVE IFLETYP WAIT PARLAM RUNREPEAT
SLAVNAME SLAVDIR
(once for each slave)
(RUNTIME(I), I=1,NSLAVE)
Any lines after this point are required only if IFLETYP is nonzero; the
following group of lines is to be repeated once for each slave.
INFLE(1)
INFLE(2)
(to NTPFLE lines, where NTPFLE is the number of template files)
OUTFLE(1)
OUTFLE(2)
(to NINSFLE lines, where NINSFLE is the number of instruction files)
```

Structure of a Parallel PEST run management file.

It is important to note that if the RUNREPEAT variable is present in a PEST run management file, then the PARLAM variable must also be present.

2.12 Covariance Matrix Files

As documented in the PEST manual, PEST is able to read an observation covariance matrix file in place of weights. Enhancements to PEST have been made in order to now allow covariance matrices to be supplied in files of two different formats. The existing format, as documented in the PEST manual, is still supported; this requires that the matrix be supplied in an ASCII file with space or comma delimited entries. In addition to this, PEST is now able to read "matrix files" whose storage protocols are described later in this document. Thus the covariance matrix used for specification of measurement uncertainty can be constructed and manipulated using PEST's new matrix handling utilities.

The following should be noted:-

1. PEST detects itself whether a matrix is supplied using the old format, or the newer matrix file format.
2. The matrix supplied to PEST must be square and symmetrical. It must possess the same number of rows and columns as there are observations in the observation group to which it pertains.
3. Matrix file protocol requires that rows and columns be named. PEST does not check these names against the names of observations comprising the observation group to which the matrix is assigned. Rows and columns in the covariance matrix must thus be supplied in the same order as that in which associated observation or prior information names are listed in the PEST control file. That is, items in the covariance matrix file are linked by order or occurrence and not by name.
4. The matrix itself (or just its diagonal elements if the ICODE variable is supplied as -1), is read using free field format.
5. If any errors are detected within the matrix file, PEST ceases execution with an appropriate error message.

2.13 New Automatic User Intervention Default Settings

As is described in the PEST manual, automatic user intervention can be implemented simply by adding the string "aui" to the eighth line of the PEST control file, just after the PHIREDSWH variable. As well as this, if desired, an "automatic user intervention" section can be added to the PEST control file in which values are assigned to variables which govern the operation of PEST's automatic user intervention functionality. If this section is not added, PEST assigns default values to these control variables.

Default values for automatic user intervention control variables are now as set out in the table below.

Variable name	**Default value**
MAXAUI	0.75 times number of adjustable parameters
AUISTARTOPT	1
NOAUIPHIRAT	0.9
AUIRESTITN	0
AUISENSRAT	5.0
AUIHOLDMAXCHG	0
AUINUMFREE	3
AUIPHIRATSUF	0.8
AUIPHIRATACCEPT	0.99
NAUINOACCEPT	0.75 times MAXAUI

Default settings for automatic user intervention control variables.

The settings supplied in the above table are such as to cause PEST to make every effort possible to lower the objective function on every iteration. Hence computation times may be greatly lengthened (especially as the automatic user intervention process is not parallelisable). Hence, in certain circumstances, the user may wish to supply different values for these variables in order to reduce the number of model runs required per optimisation iteration.

2.14 Inclusion of Predictive Noise in Predictive Analysis Process

2.14.1 General

Authors such as Vecchia and Cooley (1987) discuss the difference between a predictive confidence interval and a prediction interval. The former is the "wiggle room" in a model prediction that is allowed by the presence of measurement noise in data employed in the model calibration process. This is the only interval calculated by previous versions of PEST's predictive analyser in which a user-specified prediction is maximised or minimised while maintaining the model in a "calibrated state" at a certain level of confidence. The means by which this level of confidence is calculated is presented below.

Unfortunately, the predictive confidence interval takes no account of the fact that a model

cannot replicate the operation of an environmental system exactly. Associated with every prediction that a model makes is a certain degree of "predictive noise". Part of this noise (probably the greater part) is the result of model inadequacies. Part of it is attributable to the fact that if a field measurement were made corresponding to the model prediction, then a certain amount of noise would be associated with that measurement. In establishing an interval about a model prediction that would encompasses this actual measurement at a certain level of confidence, this noise must be taken into account.

Where a prediction is of the same or similar type to measurements employed in the calibration process, the variance of predictive noise (if it can be assumed to be the same as that of measurement noise) is estimated through the calibration process. Suppose that the minimised objective function obtained through overdetermined inversion is Φ_{min}. Suppose also that each measurement weight employed in the inversion process is proportional to the inverse of the standard deviation of the measurement to which it pertains. PEST calculates a reference variance as:

$$\sigma_r^2 = \Phi_{min}/(n\text{-}m) \qquad (2.1)$$

where n is the number of observations on which calibration is based and m is the number of estimated parameters. If the weight assigned to the i'th observation is w_i, then its standard deviation is thus estimated as:

$$\sigma_i = (\sqrt{\sigma_r^2})/w_i \qquad (2.2)$$

Where a prediction is of an entirely different type to measurements employed in the calibration process, then the level of noise associated with that prediction (in most cases resulting from incapacity of the model to simulate all details of system behaviour which pertain to this prediction) must be guessed.

2.14.2 One Methodology for Accommodation of Predictive Noise

One way in which the presence of predictive noise can be accommodated in the predictive uncertainty analysis process is to add an extra parameter, *e* to the existing PEST parameter set prior to undertaking that process. This parameter is actually the "predictive error" associated with the prediction that is to be maximised or minimised. The "estimated value" for *e* (which would normally be supplied as its initial value), is zero. A prior information equation would then be added to the inversion process in which *e* was equated to zero. The weight assigned to this prior information equation would be such that when its inverse was multiplied by the square root of the reference variance, the standard deviation of predictive noise is obtained. Or, looking at it another way, the inverse of the weight assigned to this prior information equation should bear the same relationship to predictive noise standard deviation as the inverse of observation weights employed in the calibration process bear to their respective measurement errors.

A new template file would then be prepared which would allow PEST to record the current value of *e* prior to each model run. Alternatively, the value of *e* could be written to an existing model input file using a slightly modified existing template file; the model would then need to be modified to read this value. Functionality would then be added to the model (or to a model postprocessor) through which *e* was added to the model-calculated prediction. This new noise-augmented prediction would then be read by PEST as the quantity to be maximized or minimized through the predictive analysis process. In maximising/minimising the prediction within the objective function constraints allowed to it, PEST would then have the choice of increasing/decreasing the actual model output corresponding to this prediction,

or the error associated with this prediction or, more likely, a combination of the two.

2.14.3 An Alternative Methodology

The constrained maximization/minimization algorithm which underpins PEST's predictive analyser has been enhanced so that the same outcomes as the above process can be achieved without making significant modifications to the PEST predictive analysis input dataset, and without needing to augment the model's functionality such that it adds noise to the selected predictive output. This methodology is based on theory presented in Vecchia and Cooley (1987) for achieving this aim.

Suppose that a prediction *s* is calculated from model parameters encapsulated in the vector **p** using the linearised model equation:-

$$s = \mathbf{y}^t\mathbf{p} \tag{2.3}$$

where **y** is a vector of sensitivities of the model prediction to each of the parameters encapsulated in **p**; predictive and parameter offsets are ignored in this and following equations. Suppose that we wish to maximize the model output corresponding to this prediction while constraining the objective function (which, for the moment, will be assumed NOT to include a contribution from predictive noise *e*) to be no higher than Φ_0. Then **p** corresponding to the maximised/minimised prediction can be shown to be calculable as:-

$$\mathbf{p} = \left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\left\{\mathbf{X}^t\mathbf{Q}\mathbf{h} - \frac{\mathbf{y}}{2\lambda}\right\} \tag{2.4}$$

where:-

$$\left(\frac{1}{2\lambda}\right)^2 = \pm\frac{\Phi_0 - \mathbf{h}^t\mathbf{Q}\mathbf{h} + \mathbf{h}^t\mathbf{Q}\mathbf{X}\left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{h}}{\mathbf{y}^t\left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\mathbf{y}} \tag{2.5}$$

and the **h** vector is comprised of observations employed in the calibration process (once again with offsets ignored). The positive or negative square root of $(1/2\lambda)^2$ is chosen depending on whether prediction maximization/minimization is being performed. These are the same equations as those presented in Chapter 2 of the PEST manual.

Where Φ_0 includes the effect of predictive error (and therefore includes the term $(w_e e)^2$ where w_e is the weight assigned to predictive noise and *e* is the predictive error itself) equation (2.4) is still employed to calculate **p**. However (2.5) becomes:-

$$\left(\frac{1}{2\lambda}\right)^2 = \pm\frac{\Phi_0 - \mathbf{h}^t\mathbf{Q}\mathbf{h} + \mathbf{h}^t\mathbf{Q}\mathbf{X}\left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{h}}{\mathbf{y}^t\left(\mathbf{X}^t\mathbf{Q}\mathbf{X}\right)^{-1}\mathbf{y} + w_e^{-2}} \tag{2.6}$$

while the actual predictive error *e* is calculated as:-

$$e = -w_e^{-2}/2\lambda \tag{2.7}$$

For nonlinear models equations 2.4 to 2.7 are solved for **p** using an iterative procedure in which **X**, **y** and **h** (which is actually replaced by the residuals vector **r**) are updated during every optimisation iteration in accordance with current values of **p**.

2.14.4 Objective Function Constraints

Vechhia and Cooley (1987) and Christensen and Cooley (1999) discuss the difference between simultaneous (or Scheffé) confidence/prediction intervals and individual confidence/prediction intervals. A *simultaneous* parameter confidence region is an *m*-dimensional region in parameter space bounded by a surface of constant objective function value (which is an ellipsoid for a linear model) within which, at a nominated level of confidence, all parameters lie. The projections of this surface onto different parameter axes define the Scheffé confidence intervals for these parameters. The Scheffé 1-α confidence interval for a prediction is defined as the interval between the maximum and minimum prediction that can be made by the model using parameters that lie within the joint parameter confidence region pertaining to a nominated 1-α probability level. The simultaneous prediction interval includes predictive noise as a notional parameter in this process, and as an augmenter of the prediction, in the manner described above.

The 1-α *individual* confidence interval of a parameter is simply the range of values, centred on the best parameter estimate, within which the parameter lies at a confidence limit of 1-α, irrespective of values taken by other parameters. The 1-α individual confidence limit of a prediction is defined as the range of predictive values for which the combination of parameters that gives rise to the prediction is within the 1-α confidence interval for that combination of parameters. Individual predictive intervals take predictive error into account as both a notional parameter and a prediction augmenter as discussed above. Individual confidence and predictive intervals are smaller than simultaneous intervals at the same level of confidence.

Values of Φ_0 for 1-α simultaneous confidence and prediction intervals respectively are given by:-

$$\Phi_0 = \Phi_{\min}\left[\frac{m}{n-m}F_\alpha(m,n-m)+1\right] \qquad (2.8)$$

and

$$\Phi_0 = \Phi_{\min}\left[\frac{m+1}{n-m}F_\alpha(m+1,n-m)+1\right] \qquad (2.9)$$

Where "*F*(*m*,*n*-*m*)" refers to the F distribution with (*m*,*n*-*m*) degrees of freedom and $\Phi_{\min}$ is the minimized objective function as calculated during the calibration process (which must precede the predictive analysis process).

For individual (1-α) predictive confidence and prediction intervals respectively, the following equations apply:-

$$\Phi_0 = \Phi_{\min}\left[\frac{t^2_{\alpha/2}(n-m)}{(n-m)}+1\right] \qquad (2.10)$$

and

$$\Phi_0 = \Phi_{\min}\left[\frac{t^2_{\alpha/2}(n-m)}{(n-m)}+1\right] \qquad (2.11)$$

Where *t*(*m*-*n*) signifies a *t* distribution with (*n*-*m*) degrees of freedom. Obviously equation (2.11) is identical to (2.10). However in the latter case the predictive error term $(ew_e)^2$ is

included in the objective function Φ (as it is in equation 2.9); note however that its contribution to Φ_{min} is zero due to the fact that *e* is zero at the calibration objective function minimum. Christensen and Cooley (1999) and Cooley (2004) point out that a correction factor should be applied to the right side of equations 2.10 and 2.11 to accommodate nonlinear model behaviour. However they also point out that this factor is mostly close to 1.0; hence it will be ignored in the present work.

2.14.5 Implementation in PEST

Calculation of the confidence interval for a particular prediction is simply a matter of using PEST's predictive analyzer as described in the PEST manual, and employing Φ_0 of equation (2.8) or equation (2.10) as the PD0 predictive analysis control variable.

Calculation of prediction intervals is only slightly more complex. In this case Φ_0 of equation (2.9) or equation (2.11) must be employed for PD0. Furthermore, the weight assigned to the observation comprising the sole member of the observation group "predict" must be correct in terms of its characterisation of predictive noise in the current calibration context, for PEST does not ignore this weight when calculating prediction intervals as it does when calculating confidence intervals, instead adopting it as w_e in the above equations. As discussed above, this weight must bear the same relationship to predictive noise as observation weights bear to observation noise. Finally, a new PEST control variable named PREDNOISE must be set to 1. The location of the PREDNOISE variable in the "predictive analysis" section of the PEST control file is illustrated in the figure below.

```
* predictive analysis
NPREDMAXMIN PREDNOISE
PD0 PD1 PD2
ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH
ABSPREDSWH RELPREDSWH
NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP
```

Predictive analysis section of the PEST control file showing the location of the PREDNOISE variable.

If PREDNOISE is supplied as zero (or not supplied at all), PEST will not take predictive error into account when maximising the user-specified prediction. However if it is set to 1, then PEST's behaviour in conducting the predictive analysis maximisation/minimisation process is slightly altered. In particular:-

1. The reported objective function is now the normal model-to-measurement misfit objective function plus the contribution from predictive error, i.e. $(ew_e)^2$.
2. The reported prediction is now the model output specified as the prediction, plus the predictive noise term *e*.
3. The current value of *e* is reported with the prediction as the optimisation process progresses.

In all other respects, operation of PEST is outwardly the same when PREDNOISE is set to 1 as it is when PREDNOISE is set to zero.

Irrespective of the PREDNOISE setting serious consideration should be given to employing a line search in the predictive analysis process (not withstanding its high cost in terms of model runs). Variables controlling this line search procedure should be assigned in accordance with the discussion presented earlier in this addendum (see Section 2.4).

Limited experience to date in comparing the performance of PEST's internal accommodation

of the predictive noise term with the more lengthy procedure described in Section 2.13.2, suggests paradoxically, that the latter is slightly more efficient, and provides slightly higher/lower maximised/minimised predictions. It is thought that this is due to better accommodation of the nonlinear nature of the problem, particularly in implementing the line search, when the *e* parameter is re-calculated at each stage of this search, instead of being re-calculated only when a different Marquardt lambda is employed in the parameter calculation process as occurs for internal implementation of this procedure. However, fortunately, differences between the two outcomes do not appear to be great.

2.15 Starting a New PEST Run Using and Old Jacobian

There are many occasions on which a PEST run is commenced, only for the user to realise as soon as the first parameter upgrade is attempted, that different settings should have been employed for that run. For example, a PEST control variable should have been set differently; or perhaps PEST should have been asked to use truncated singular value decomposition as a regularisation device instead of Tikhonov regularisation. On such situations the PEST run is then halted before the second optimisation iteration begins, or at an early stage of this iteration where parameter sensitivities are being calculated through repeated model runs; the PEST control file is then edited, and PEST is re-started. Unfortunately, this is a wasteful procedure, for then PEST calculates parameter sensitivities for initial parameter values all over again (assuming that these have not been changed) before recalculating parameter upgrades on the basis of the new inversion settings.

Re-calculation of sensitivities for initial parameter values can now be avoided if PEST is re-started using the "/i" switch. In this case PEST immediately prompts for the name of a JCO file (i.e. an unformatted file in which the Jacobian matrix is stored – see the PEST manual for further details), from which it can read initial parameter sensitivities. When the time comes to calculate the first set of parameter derivatives, PEST then reads them from this file instead of calculating them by finite differences or reading them from a derivatives file. For all subsequent iterations, however, it calculates sensitivities in the normal manner.

If a PEST run has been halted after calculation of the first Jacobian matrix, it is a good idea to copy the JCO file to a file of another name. This will prevent it from being destroyed or overwritten next time PEST is run on the same case. The file is thus always available for use during the first iteration of subsequent PEST runs. The usefulness of this file as a powerful means to save repetition of the first inversion iteration is heightened by the fact that subsequent PEST cases that can use it for this purpose need not employ the same prior information equations as were used (or not) in the initial PEST run on whose basis the initial JCO file was stored, for PEST always calculates prior information sensitivities internally. All that is required is that the new PEST run employ the same adjustable parameters (in the same order) and the same observations (in the same order) as the old PEST run; PEST will soon inform you if this is not the case. However it is also important that the log-transformation status of parameters does not change between PEST runs; this is a condition that PEST cannot detect, and so the user must ensure that this is the case him/herself. However observations weights can change between the PEST run that produced the stored JCO file, and any subsequent run that uses it.

To start a new PEST run using an old Jacobian matrix for the initial iteration, run PEST using the command:-

```
pest case.pst /i
```

where ***case*** is the filename base of the current PEST case. As soon as it commences execution, PEST will prompt:-

```
Enter name of JCO file for first iteration sensitivities:
```

to which you should respond with the name of a JCO file produced during (normally the first iteration) of a previous PEST run. As stated above, while the name of this file can be ***case.jco*** where ***case*** is the filename base of the current PEST case, this is a very dangerous procedure due to the likelihood of this file being destroyed or overwritten. It is far better to rename a JCO file that is to be re-used for initialisation of subsequent PEST runs in this manner.

2.16 Partial Parallelisation of the Lamdba Search

The figure below shows the structure of the parallel PEST run management file.

```
prf
NSLAVE IFLETYP WAIT PARLAM
SLAVNAME SLAVDIR
(once for each slave)
(RUNTIME(I), I=1,NSLAVE)
Any lines after this point are required only if IFLETYP is nonzero; the
following group of lines is to be repeated once for each slave.
INFLE(1)
INFLE(2)
(to NTPFLE lines, where NTPFLE is the number of template files)
OUTFLE(1)
OUTFLE(2)
(to NINSFLE lines, where NINSFLE is the number of instruction files)
```

Structure of the Parallel PEST run management file.

The last variable on the second line of this file informs PEST whether or not it should partially parallelise the procedure through which it tests the effects of different Marquardt lambdas in lowering the objective function. If this is set to zero, or omitted, partial parallelisation of the lambda search is disabled. If it is set to 1, the partial parallelisation process is enabled.

Where the number of slaves involved in the parallelisation process is high, it is wise to limit the number of these slaves which actually become involved in the Marquardt lambda testing procedure. One reason for this is that, with so many different lambdas being simultaneously tested, it is possible for some to yield upgrade vectors containing parameters which create problems for the model. In general, the further is a tested lambda from the current lambda, the more likely is this to happen. Problems could include slower model convergence or even model failure. Fortunately this can now be easily prevented.

PARLAM can now be set to a negative number. As long as it is non-zero, partial parallelisation of the lambda search procedure will be activated within PEST. However the number of slaves involved in this lambda search procedure will be limited to the absolute value of PARLAM. Thus, for example, if parallelisation is being undertaken using 50 slaves, and if PARLAM is set to -4, then only 4 slaves will be involved in the partial parallelisation of lambda testing.

A PARLAM setting of -9999 however has special significance. This setting should only be employed where a user has access to a relatively large number of computing nodes of equal

power. If PARLAM is set to -9999, the parallel lambda search procedure follows the strategy set out below.

1. Only one cycle of parallel runs is devoted to testing Marquardt lambdas; PEST will not commit to a second cycle, irrespective of results forthcoming from the first cycle.
2. The maximum number of parallel runs constituting this cycle is set to NUMLAM, the variable in the "control data" section of the PEST control file which sets the maximum number of lambda-testing runs undertaken per iteration. However if there are less slaves than NUMLAM available at the time that lambda testing is required, NUMLAM is temporarily reduced to the number of available slaves.
3. If it is not supplied as a negative number, RLAMFAC (the lambda multiplier employed in lambda testing) is set internally to the negative of its supplied value. Hence, as described in section 2.18 of this addendum, a much wider range of lambda values is tested than would be the case if a value-independent multiplier were employed.
4. Tested lambda values (on different slave machines) are disposed in equal numbers above and below the current value of the Marquardt lambda.
5. The outcomes of all such runs are read, irrespective of "trends" discovered from the processing of early returns. Thus if the dependence of the objective function on the Marquardt lambda is irregular (such as can occur if finite-difference derivatives are compromised as a result of model output granularity), an "accidental" low objective function corresponding to a random lambda will not be missed.

Use of a PARLAM setting of -9999 is recommended where a user has access to a large number of nodes, many of which would be standing idle while implementing alternative lambda search strategies. The user must remember however, that the number of runs employed for lambda testing is set by NUMLAM, and not by the number of slaves/nodes available. Hence NUMLAM may need to be set higher than normal. Note also that, as stated above, only one round of lambda-testing runs is undertaken.

Caution should be exercised when employing this lambda-testing option. If a model is susceptible to crashing when it is supplied with parameter values that it finds unpalatable, this is most likely to occur during the lambda search, for this is the part of the parameter estimation process where parameter values are most different from those that have been employed before. Testing of a wide variety of lambda values in this parallel fashion can increase the chances of model run (and hence PEST) failure.

Another negative feature of this mode of lambda testing is that PEST's ability to sequentially and temporarily freeze parameters that have hit their bounds at their current values if gradient or upgrade vectors point out of bounds is compromised to some (but not a great) extent.

It is hoped that this methodology for lambda testing will be of use where, as stated above, model derivatives are compromised because of model numerical imperfections. If this is the case, the user may wish to set RLAMFAC to a higher negative value than normal (for example a value of -4 rather than the recommended value of -2) so that it can sample the objective function vs. lambda curve in greater detail (perchance to fall into objective function crevices in this noisy surface).

Note for BPROC users. The MPRUN program used to run the model is supposed to create a file named "case.number_processors" from which Parallel PEST can learn of the number of nodes that are actually employed in current parallelisation. At the time of writing, this file is

not being produced by MPRUN. Hence if NUMLAM is inadvertently set above the number of processors that are actually available, PEST has no way of knowing to reduce its value.

2.17 New NOPTMAX Setting

In the past, to run PEST solely for the purpose of calculating a Jacobian matrix, NOPTMAX had to be set to -1. With NOPTMAX thus set, as well as filling the Jacobian matrix, PEST also calculates the parameter covariance, correlation coefficient and eigenvalue matrices, as well as statistics derived from these. Residuals, and statistics based on residuals, are written to the run record file. Furthermore, a final model run is undertaken based on "optimised parameters" (in this case initial parameters) so that model input and output files remaining after the PEST run is complete pertain to these parameters.

Where sensitivities are being calculated on the basis of base parameters prior to an SVD-assisted PEST run, there is no need for these statistics to be calculated. Not only is the user unlikely to be interested in them, but where there are many base parameters their computation (particularly those of the covariance and related matrices) may take a considerable amount of time. Calculation of these can now be avoided through setting NOPTMAX to -2 instead of -1. In this case, PEST terminates execution as soon as the Jacobian matrix has been calculated and recorded in the pertinent JCO file.

2.18 New RLAMFAC Setting

Operation of the Marquardt lambda is governed by variables which appear on the sixth line of the PEST control file. In normal operation, during each optimisation iteration, the Marquardt lambda is varied either up or down (or both if necessary) in search of a value that provides the largest reduction in the objective function (unless the initial Marquardt lambda is set to zero, in which case it is not varied from this value during the entirety of a PEST run). In undertaking these variations, PEST multiplies or divides the Marquardt lambda by a factor, the value for which is supplied as the RLAMFAC variable. This variable must always have a value greater than 1.0.

In some instances it is apparent that when the Marquardt lambda is very high, it may need to be varied by more than the same factor by which it is varied when it is low. This has been found especially when undertaking parameter estimation with SVDMODE set to 2 (see below). In this case the Marquardt lambda serves as a regularisation variable which is added to singular values rather than normalised and then added to the diagonal terms of the $\mathbf{X}^t\mathbf{QX}$ matrix. Limited experience suggests that when it is required for stabilization of the inverse problem, it sometimes needs to be high – so high that it dominates the singular values to which it is added. When it is then varied in an attempt to lower the objective function as much as possible, it needs to be varied by a large amount to have any effect on the calculation of new parameters.

It has also been found to be useful in conjunction with PEST's AUI functionality, and on other occasions where ill-posedness of the inverse problem requires that the Marquardt lambda be rapidly raised in order to promote lowering of the objective function.

In response to this requirement, a new methodology is now offered for Marquardt lambda adjustment. This is activated by setting the value for RLAMFAC to a negative number – a number whose absolute value must be greater than 1.0 (just as in the case of standard operation of the Marquardt lambda). When set to a negative value in this way, the Marquardt lambda adjustment factor *f* is re-calculated during every iteration of the parameter estimation

process as follows:-

$f = \min[\lambda^{1/r}, 2.0]$ if $\lambda > 1.0$

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

$f = 2.0$ if $\lambda = 1.0$

where λ is the value of the Marquardt lambda at the beginning of a particular optimisation iteration and r is the absolute value of the user-supplied RLAMFAC variable. Operation of this variable is easily pictured by the fact that, whether λ begins an iteration with its value above or below 1.0, its value will have reached 1.0 after r multiplications or divisions by RLAMFAC (provided this is the direction in which PEST decides to move it). Thus, for example if RLAMFAC as supplied in the "control data" section of the PEST control file is set to -2.0 and the value of λ is 1000.0 at the start of a particular optimisation iteration, the Marquardt lambda adjustment factor is temporarily set to 31.62 for that iteration. Note, however that, as indicated in the above equations, it will never be set lower than 2.0.

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

2.19 New Setting for SVDMODE

SVDMODE in the "singular value decomposition" section of the PEST control file can now be set to 2 as well as 1; in both cases solution of the inverse problem takes place through singular value decomposition. For a setting of 2, singular value decomposition takes place on the basis of $\mathbf{Q}^{1/2}\mathbf{X}$ rather than $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ (as it does for a setting of 1). This is much faster in cases where parameters outnumber observations (and even in cases of large numbers of observations where formation of the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix is a numerically tedious task).

Setting SVDMODE to 2 has a number of other repercussions. These are now outlined.

1. On the basis of limited experience, it has been found that where SVD is employed for solution of an underdetermined inverse problem, some kind of solution damping is required in addition to the regularizing effects of SVD, especially where there are many parameters. In normal PEST operation (and with SVDMODE set to 1) this task is accomplished using the Marquardt lambda, whereby a scaled diagonal term is added to the diagonal of the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix before it is inverted (or decomposed). With SVDMODE set to 2 damping is still undertaken and, as far as PEST output is concerned, the damping parameter is still assigned the term "Marquardt lambda". However this damping variable cannot be added to the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix, because this matrix is never formed. Instead it is added to all singular values of $\mathbf{Q}^{1/2}\mathbf{X}$ before solving for the parameter upgrade vector. Based on limited testing, this strategy appears to work well. However it should be noted that the damping parameter is not normalized before addition to these singular values, and hence its value is not directly comparable between one PEST run and another. (Normalisation with respect to the highest singular value was attempted, but did not seem to work well.) Providing RLAMFAC with a negative value (see the preceding section) often assists in the efficiency of finding an optimal damping variable at any given stage of the parameter estimation process as it allows rapid variation of the Marquardt lambda during any optimisation iteration in search of its optimum value.

2. If SVDMODE is set to 2, memory conservation is automatically activated. Hence the covariance matrix, and matrices derived from it, are not listed at the bottom of the PEST control file. Nor are they written to the MTT file.

3. As for an SVDMODE setting of 1, PEST records an abundance of information to the "SVD file", the amount of this information being governed by the setting of the EIGWRITE variable. If this is set to zero, eigenvectors are not recorded; if it is set to 1, eigenvectors are written to this file. However whereas the number of eigenvectors recorded in this file is equal to the number of adjustable parameters when SVDMODE is set to 1, if the number of observations plus prior information equations is smaller than this, then the number of eigenvalues/eigenvectors recorded in the SVD file is restricted to the total number of observations plus prior information equations. (This is also the maximum number of dimensions that the parameter solution space can hold.)

2.20 Using LSQR instead of Truncated SVD

The LSQR algorithm (Paige and Saunders; 1982a, 1982b) can be employed to implement regularised inversion when running in "parameter estimation" mode, just as truncated SVD can be employed to do the same thing. This is done by introducing an "lsqr" section to the PEST control file in place of (or directly underneath) a "singular value decomposition" section. An example of an "lsqr" section is shown below. The names of the PEST variables appearing in it are shown under that.

```
* lsqr
1
1e-10  1e-10  1e3  10000
1
```

Example of an LSQR section of a PEST control file

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

Variables used in the LSQR section of the PEST control file.

The role of each of the variables appearing in the "lsqr" section of the PEST control file is now described.

LSQRMODE

This must be set to 0 or 1. If it is set to 1, LSQR is employed for solution of the inverse problem. The LSQR algorithm is applied to the matrix $\mathbf{Q}^{1/2}\mathbf{X}$ where $\mathbf{X}$ is the Jacobian matrix.

LSQR_ATOL

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

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

LSQR_BTOL

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

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

LSQR_CONLIM

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

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

LSQR_ITNLIM

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

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

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

LSQRWRITE

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

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

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

1. *Unsymmetric equations - solve* **Ax** = **b**;
2. *Linear least squares - solve* **Ax** = **b** *in the least-squares sense;*

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

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

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

Note the following.

1. PEST will not allow LSQR to be employed for solution of an inverse problem if automatic user intervention is activated, or if PEST is run in "predictive analysis" mode.
2. PEST permits a "singular value decomposition" section and an "lsqr" section to exist in the same PEST control file. However it does not permit both of SVDMODE and LSQRMODE to have non-zero values; but it does allow both of them to be simultaneously set to zero.
3. In setting values for LSQR control variables, do not forget that the numerical accuracy of terms of the **A** matrix is set by the accuracy of derivatives calculation. This is determined by the model if derivatives are supplied externally. However if finite differences are employed for calculation of derivatives, the numerical precision of the elements of **A** will probably be of the order of 10^{-2} or 10^{-3} at best.
4. Do not forget that observations through which parameters are inferred are contaminated by measurement noise. This has repercussions for the setting of the *lsqr_btol* and *lsqr_conlim* variables. In short, the more contaminated by noise is the observation dataset, the fewer LSQR iterations may be suitable for solution of an inverse problem.
5. The user is not given the opportunity to supply a value for the LSQR *damp* variable. Rather this is supplied to the LSQR subroutine as the current value of the Marquardt lambda. Its value is reported to the screen and to the PEST run record file. Variables which control is value can be supplied in the "control data" section of the PEST control file in the usual manner.
6. If inversion is undertaken using LSQR, PEST memory conservation is automatically implemented. Thus items such as the parameter covariance matrix (which is not of much use in the regularised inversion context) are not listed to the run record file, the matrix (MTT) file is not written, etc.

2.21 Greater Flexibility of Compressed Matrix Storage

If the MAXCOMPDIM variable in the "control data" section of the PEST control file is assigned a value greater than 1, PEST stores the Jacobian matrix in compressed form. Using this storage format, zero-valued elements are omitted. This can result in the ability to hold a matrix of far larger dimensions than would otherwise be possible, especially where a long series of prior information equations citing only a small number of parameters is employed for regularisation purposes. A problem arises however, in knowing what value to give to MAXCOMPDIM. This informs PEST how many elements it must set aside for Jacobian

matrix storage. If this number is set too high, storage is wasted. If it is set too low, PEST ceases execution with an error message when allocated storage is full.

It is often difficult to know ahead of time how many non-zero elements the Jacobian matrix will possess. In previous versions of PEST it was necessary to re-commence execution from the very start of the parameter estimation process when this error message was encountered. Hence any model runs already undertaken in Jacobian matrix computation were wasted. In versions later than 11.0, PEST execution can be re-commenced using the "/s" or "/d" switch after altering MAXCOMPDIM upwards in the PEST control file. Thus no model runs are lost.

2.22 SVDA - Automatic Super Parameter Derivatives Computation

Under certain circumstances it is possible to eliminate the need to calculate super parameter sensitivities by finite differences on the first iteration of an SVD-assisted parameter estimation exercise. In particular, if

1. SVDA_EXTSUPER is set to a value other than 1, indicating that super parameters are calculated by PEST on the basis of a base parameter Jacobian matrix, and
2. this matrix pertains to the same observations and prior information equations as those featured in super parameter PEST control file,

then derivatives of super parameters can be calculated from base parameter sensitivities, eliminating the need for them to be calculated using finite differences. Fortunately, these are the most common conditions in which SVD-assisted parameter estimation is undertaken.

A new control variable in the "SVD-assist" section of the PEST control file activates the functionality whereby PEST is able to dispense with finite-difference calculation of super parameter sensitivities during its first iteration, calculating these from base parameter sensitivities instead. The name of this variable is SVDA_SUPDERCALC; it is the fourth variable to appear on the third line of this section (immediately following the SVDA_EXTSUPER variable (see below for a refresher on SVDA control variables). If it is omitted or set to zero, then super parameter derivatives calculation takes place through finite differences in the usual manner during the first optimisation iteration of the inversion process. However if it is set to 1, then PEST calculates super parameter derivatives internally for the first iteration of the SVD-assisted parameter estimation process, removing the necessity for any model runs to be undertaken in calculating these derivatives.

The SVDA_SUPERDERCALC variable is recorded in bold in the "SVD assist" section of the PEST control file reproduced below.

```
* svd assist
case.pst
case.jco
0    2    1    1
```

The SVDAPREP utility has been upgraded to reflect this new aspect of PEST's SVD-assist functionality. Unless the user informs SVDAPREP that super parameters are to be supplied externally, it now prompts:-

```
Automatic calc. of 1st itn. super param. derivs?  [y/n] (<Enter> if "y") :
```

If the user responds with "y" or <Enter> to this prompt, SVDA_EXTSUPER is set to 1 in the SVDAPREP-generated PEST control file. Otherwise it is set to 0.

2.23 SVDA – Super Parameter Definition through SVD

As has been explained elsewhere, the last line of the "SVD assist" section of a PEST control file contains four variables, the third of which is named SVDA_EXTSUPER. This variable informs PEST how super parameters are to be formed. If SVDA_EXTSUPER is set to 1, then super parameters are read from an external file. However if it is set to any other value, these are calculated internally by PEST on the basis of sensitivities supplied in the Jacobian matrix whose name appears on the previous line of the PEST control file (with any sensitivities pertaining to regularisation observations omitted from this calculation).

In previous versions of PEST, setting SVDA_EXTSUPER to zero caused PEST to formulate super parameters through singular value decomposition of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$, where $\mathbf{X}$ represents the base parameter Jacobian matrix contained in the nominated base Jacobian matrix file. PEST functionality is now such that these are calculated on the basis of $\mathbf{Q}^{1/2}\mathbf{X}$ instead. The result should be the same; however their computation will often be much quicker, especially where parameters outnumber observations. Also, memory requirements can be much smaller.

Should the user desire that super parameter definition take place on the basis of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$, this can be requested by setting SVDA_EXTSUPER to 3.

2.24 SVDA - Using LSQR for Super Parameter Definition

If SVDA_EXTSUPER is set to 2, PEST calculates super parameters using the first *m* "**v**" vectors computed by the LSQR algorithm, where *m* is the number of super parameters cited in the PEST control file. If SVDA_EXTSUPER is set to -2, these vectors are orthogonalised before being employed for definition of super parameters.

In cases where the number of base parameters is every high, use of the LSQR algorithm for super parameter definition may result in faster computation of these parameters.

SVDAPREP's operations have been slightly altered to reflect this new functionality (as well as that discussed in the previous section of this addendum). If the user indicates to SVDAPREP that a pre-defined super parameter file does not exist, SVDAPREP asks the user how PEST should calculate super parameters. The following options are provided:-

```
For computation of super parameters:-
   if SVD on Q^(1/2)X               - enter 1
   if SVD on XtQX                   - enter 2
   if LSQR without orthogonalisation - enter 3
   if LSQR with orthogonalisation    - enter 4
Enter your choise (<Enter> if 1):
```

These options correspond to SVDA_SUPDERCALC settings of 0, 3, 2 and -2 respectively.

2.25 SVDA - A Refresher on Control Variables

Four SVDA control variables appear on the third line of the "SVD assist" section of the PEST control file. These are (in order of appearance on this line):-

```
SVDA_MULBPA SVDA_SCALADJ SVDA_EXTSUPER SVDA_SUPDERCALC
```

For ease of reference, their roles are briefly repeated. Note that all of them are integers.

SVDA_MULBPA

SVDA_MULBPA must be set to either 0 or 1. If it is set to 1 then a series of "BPA files" is be recorded in the course of the parameter estimation process, each containing base parameter

values as estimated during subsequent iterations.

SVDA_SCALADJ

This must be set to ±4, ±3, ±2, ±1 or 0. It sets the type of parameter scaling that is undertaken in super parameter definition. Where some parameters are not log-transformed, parameter scaling is essential.

SVDA_EXTSUPER

The SVDA_EXTSUPER control variable stipulates the manner in which super parameters are computed; it must be set to 0, 1, 2, -2 or 3. If it is set to 0 PEST computes super parameters on the basis of singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$ where **X** is the user-supplied base Jacobian matrix. If it is set 1 PEST reads super parameter directions from an external file. If it is set to 2 PEST computes super parameters on the basis of "**v**" vectors computed by the LSQR algorithm. These are first orthogonalised if SVDA_EXTSUPER is set to -2. If it is set to 3 PEST computes super parameters on the basis of $\mathbf{X}^t\mathbf{QX}$.

SVDA_ SUPDERCALC

This must be 1 or 0. If it is set to 1 (and SVDA_EXTSUPER is not set to 1), PEST calculates super parameter derivatives from base parameter derivatives during the first optimisation iteration of the parameter estimation process. This saves as many model runs as there are super parameters.

2.26 SVDA – Model-Calculated Derivatives

2.26.1 General

In previous versions of PEST the use of external derivatives functionality in conjunction with SVD-assisted parameter estimation was not allowed. This was based on the fact that when undertaking SVD-assisted parameter estimation PEST is actually estimating values for super parameters (of which the model has no knowledge) while the model supplies derivatives with respect to base parameters.

Model-generated derivatives can now be employed when PEST is undertaking SVD-assisted parameter estimation, provided super parameters are not supplied externally (i.e. provided SVDA_EXTSUPER is not set to 1). When this is done PEST re-formulates super parameters during each iteration of the SVD-assisted parameter estimation process, and calculates derivatives with respect to these super parameters on the basis of the same linear combinations of base parameters as are employed for definition of the super parameters themselves.

External derivatives functionality is activated in the same way as it is for non-SVD-assisted parameter estimation. That is, the JACFILE variable on the fifth line of the PEST control file is set to 1. As well as this, the DERCOM value for all super parameters in the "parameter data" section of the PEST control file should be set to zero; thus PEST is prevented from calculating any super parameter derivatives by finite differences.

The following should be noted.

1. In normal operation, PEST allows two non-zero JACFILE settings. If JACFILE is set to 1 then derivatives are expected in PEST derivatives file format. If it is set to 2, derivatives are expected in JUPITER derivatives file format. At present, the second

option is not allowed with SVDA. This is not expected to be a disadvantage, for there is a large computational overhead in reading a JUPITER derivatives file because parameters and observations can, theoretically, be arranged in any order.

2. When used in a non-SVDA context, a value of -1.11e33 in an external derivatives file indicates that the derivative is not supplied in that file; it is then presumed that PEST calculates the derivative using finite parameter differences. This option is not available when external base parameter derivatives are supplied in the SVDA setting, because PEST has no way of calculating finite-difference derivatives for base parameters. If such a value is supplied in a derivatives file PEST will give an "out of range" error message. (I am presently minded to build an external "derivatives manager" which allows derivatives calculation to take place outside of PEST; then finite difference and model-calculated derivatives could co-exist for base parameters. This would facilitate operation of PEST with composite models, some of which can supply their own derivatives while some cannot. Derivatives calculation for the latter could then be parallelized for greater computational efficiency.)
3. When undertaking SVD-assisted parameter estimation with external derivatives, it is still necessary for a set of original base parameter derivatives to exist in a JCO file prior to the commencement of PEST execution – just as in normal SVDA operation. These derivatives are used for initial definition of super parameters, initial computation of super parameter derivatives, and for the setting of internal parameter scaling variables. While this is a little cumbersome it constitutes minimal departure from normal PEST operation; greater departures would require more profound programming changes to PEST than those which have already taken place.

2.26.2 Alterations to SVDAPREP

If the JACFILE control variable is set to 1 in a base parameter PEST control file whose name is supplied to SVDAPREP, SVDAPREP will write a PEST control file for SVD-assisted parameter estimation in which external derivatives computation is also activated. Note the following facets of SVDAPREP operation under these conditions.

1. If JACFILE is set to 1 in a base parameter PEST control file, then a "derivatives command line" section must be present within the same PEST control file.
2. Both the model command (in the "model command line" section of the base PEST control file) and the derivatives command (in the "derivatives command line" section of the base PEST control file) must be the names of batch (on a PC) or script (on a UNIX platform) files. In the former case they must possess the extension "*.bat*".
3. In normal SVDAPREP operation the model batch file is supplemented with SVDAPREP-generated commands to run PARCALC and (if there is prior information present in the base PEST control file) PICALC. Commands are also added to delete PARCALC-generated model input files. In doing this, SVDAPREP writes a new model batch file named *svdabatch.bat*. The contents of the derivatives batch file are supplemented in a similar manner; the new derivatives batch file is named *svdabatch_d.bat*.
4. Irrespective of whether or not a "derivatives command line" section is present in the base PEST control file, if JACFILE is set to zero in this file, the DERCOM variable for all super parameters is set to 1 in the SVDAPREP-generated super parameter PEST control file, thereby instructing PEST to compute derivatives for all super

parameters using finite differences. Alternatively, if JACFILE is set to 1 in the base parameter PEST control file, SVDAPREP supplies a DERCOM value of zero to all super parameters, thus requiring that the model supply derivatives for all of them. In fact, as discussed above, the model provides base parameter derivatives and PEST computes super parameter derivatives from these.

5. As noted above, if base parameter derivatives are computed externally, PEST re-defines super parameters during every optimisation iteration, and computes super parameter derivatives from base parameter derivatives. Thus PEST effectively sets the SVDA_SUPDERCALC control variable to 1 internally, whether or not it is set to this value in the base parameter PEST control file. SVDAPREP does not therefore ask the user whether super parameter derivatives are to be calculated automatically for the first PEST optimisation iteration if JACFILE is set to 1 in the PEST control file to which it is directed, for this question is irrelevant under these conditions.

There is a nuance of SVDAPREP operation of which the user should be aware. If JACFILE is set to 1, then prior information cannot be added to a base parameter PEST control file after the JCO file has been written and before SVDAPREP is run. (This is a relatively unusual situation.) If JACFILE is set to 1, PEST requires that prior information derivatives actually be stored in the JCO file. The same applies if SVDA_SUPDERCALC is set to 1. However where super parameter derivatives are computed using finite differences, this is not required.

2.27 Broyden's Jacobian Matrix Update

2.27.1 General

I'd like to thank Brian Skahill from USACE, Vicksburg and Jeff Baggett from University of Wisconsin, La Crosse for bringing this methodology to my notice. At the time of writing, limited testing has shown that this technique can increase PEST efficiency a great deal in all of its modes (i.e. estimation, predictive analysis and regularisation modes). Therefore, it is recommended that it be implemented as a matter of course. The method is briefly described below. See also:-

Madsen, K. and H.B. Nielsen, 2004. Methods for Non-Linear Least Squares. Informatics and Mathematical Modelling. Technical University of Denmark.

Skahill, B.E., and Baggett, J.S. (2006). More Efficient Derivative-based Watershed Model Calibration, Eos Trans. AGU, 87(52), Fall Meet. Suppl., Abstract 1521.

2.27.2 Description

During every iteration of the parameter estimation process, PEST computes derivatives of model outputs with respect to all adjustable parameters. It then computes upgrades for parameter values, and tests the efficacy of these upgrades by running the model using these trial parameters. If the objective function is lowered the upgrade is provisionally accepted. Further upgrade attempts are usually made, normally on the basis of extra values of the Marquardt lambda.

If a model is linear, PEST should be able to reduce the objective function to zero in just one upgrade attempt of one iteration. But rarely is this the case, because derivatives that are computed at one location in parameter space are not quite applicable to other locations in parameter space. Thus after a suitable parameter upgrade vector has been selected at one location in parameter space (this resulting in a lowered objective function), derivatives are re-

computed at the upgraded location, and another optimisation iteration is undertaken.

On any occasion that a parameter upgrade is attempted, there is an opportunity to improve the current set of parameter derivatives encapsulated in the Jacobian matrix, simply on the basis of the information that has been obtained from that upgrade attempt. The upgrade vector defines a secant in parameter space, the slope of which can be employed to improve the components of parameter derivatives projected onto that direction. After undergoing revision on the basis of this new information, the updated Jacobian matrix can be employed for computation of subsequent upgrade vectors, thereby hopefully making these subsequent parameter upgrade attempts (computed on the basis of different Marquardt lambdas) more effective. When upgrade attempts have been exhausted, another optimisation iteration is commenced and all parameter derivatives are re-computed on the basis of the upgraded parameter set as a new optimisation iteration is commenced.

Let **J** be the current Jacobian matrix computed on the basis of the current parameter set **p**. Suppose that a trial set of parameters $\mathbf{p}_{new}$ has just been computed on the basis of this Jacobian matrix in conjunction with a particular Marquardt lambda. Define the upgrade vector **u** as:

$$\mathbf{u} = \mathbf{p}_{new} - \mathbf{p}$$

Let the vectors $\mathbf{o}_{new}$ and **o** represent model outputs (corresponding to field observations) computed on the basis of $\mathbf{p}_{new}$ and **p** respectively. Implementation of Broyden's rank one update process results in the production of a new Jacobian matrix $\mathbf{J}_{new}$, calculated as follows:-

$$\mathbf{J}_{new} = \mathbf{J} + \mathbf{w}\mathbf{u}^t$$

where

$$\mathbf{w} = \frac{1}{\mathbf{u}^t\mathbf{u}}\left(\mathbf{o}_{new} - \mathbf{o} - \mathbf{J}\mathbf{u}\right)$$

$\mathbf{J}_{new}$ is then employed in computation of further parameter upgrade attempts made during the current optimisation iteration using the theory presented in chapter 2 of the PEST manual.

2.27.3 Implementation of Broyden's Update

The arrangement of variables employed in the "control data" section of the PEST control file is shown below (with optional variables placed in square brackets). A new variable named JACUPDATE (shown in bold below) controls the operation of the Broyden Jacobian update procedure in PEST.

```
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM]
NTPLFLE NINSFLE PRECIS DPOINT NUMCOM JACFILE MESSFILE
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND]
PHIREDSWH [NOPTSWITCH] [DOAUI]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR
ICOV ICOR IEIG
```

Variables in the "control data" section of the PEST control file.

If JACUPDATE is omitted or set to zero, Broyden updating will not occur. If it is set to *n*

where *n* is a positive integer, it will occur immediately following the first *n* parameter upgrade attempts made during every optimisation iteration on the basis of different Marquardt lambdas. If it is not set to zero, it is suggested that **JACUPDATE be set to 999** so that the Jacobian matrix is updated after every parameter upgrade attempt.

Limited experience to date has shown that Broyden Jacobian updating works well in most parameter estimation contexts, including all modes of PEST behaviour, and whether or not devices such as "automatic user intervention" are being employed in difficult parameter estimation contexts. However its performance does not appear to be as beneficial where singular value decomposition or LSQR is employed as a solution device. On the other hand, it appears to work well with the SVD-assist solution methodology. Feedback on this would be welcome.

2.28 Parameter Sensitivity Re-Use

2.28.1 General

If a model takes a long time to run, any opportunities to dispense with unnecessary model runs should be taken. PEST's sensitivity re-use functionality provides such an opportunity.

If, during one particular iteration of the (regularized) parameter estimation or predictive analysis process undertaken by PEST, PEST detects that the composite sensitivities of certain parameters are low, then it can be instructed not to waste a model run (or perhaps two model runs if central derivatives computation is employed) in computing sensitivities with respect to that parameter on the next occasion that the Jacobian matrix must be filled. This can occur if PEST's sensitivity re-use functionality is activated. However, no matter what the sensitivity distribution of parameters, sensitivities for all parameters will be re-computed every *N* optimisation iterations, where *N* is set by the user. (Note that Broyden updating, if activated, will still incur parameter sensitivity improvements for all parameters every iteration, regardless of the sensitivity re-use setting.)

2.28.2 Activating Sensitivity Re-Use

Sensitivity re-use is activated by setting the DOSENREUSE variable on the eighth line of the PEST control file to "senreuse". If this is set to "nosenreuse", or simply omitted, then sensitivity re-use will not be activated.

The location of the DOSENREUSE variable within the "control data" section of a PEST control file is shown below.

```
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM]
NTPLFLE NINSFLE PRECIS DPOINT NUMCOM JACFILE MESSFILE
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND]
PHIREDSWH [NOPTSWITCH] [DOAUI] [DOSENREUSE]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR
ICOV ICOR IEIG
```

Variables in the "control data" section of the PEST control file.

The following should be noted.

1. Any or all of the NOPTSWITCH, DOAUI and DOSENREUSE variables can be omitted from the eighth line of the PEST control file.
2. The DOAUI and DOSENREUSE variables can be placed in either order. However if the NOPTSWITCH variable is present, it must precede them.

2.28.3 Sensitivity Re-Use Variables

A number of variables govern implementation of PEST's sensitivity re-use functionality. These can be supplied in a special "sensitivity reuse" section. Like the "automatic user intervention" section of the PEST control file, this section can be omitted if desired. In that case PEST employs default values for sensitivity re-use variables.

If present, the "sensitivity reuse" section must be placed just before the "parameter groups" section. If a "singular value decomposition" section is present, the location of this section with respect to the "sensitivity reuse" section does not matter; however each of these must precede an "automatic user intervention" section if the latter is present in a PEST control file. They must also precede an "SVD assist" section if this is present.

Irrespective of whether a "sensitivity reuse" section is present in the PEST control file, sensitivity re-use functionality will not be activated unless DOSENREUSE in the "control data" section of the PEST control file is present, and set to "senreuse".

The arrangement of variables within the "sensitivity reuse" section of a PEST control file is shown below. Following that is an example of such a section showing default values for the variables governing its operation.

```
* sensitivity reuse
SENRELTHRESH  SENMAXREUSE
SENALLCALCINT  SENPREDWEIGHT  SENPIEXCLUDE
```

Arrangement of variables within the "sensitivity reuse" section of a PEST control file.

```
* sensitivity reuse
0.15  -1
3  -1.0  yes
```

An example of a "sensitivity reuse" section, showing default values for sensitivity re-use variables.

The roles of each of these variables is now discussed in detail.

SENRELTHRESH

This denotes the threshold of parameter composite sensitivity (relative to the maximum composite sensitivity of any parameter) at which sensitivity reuse is activated for any parameter. That is, if the composite sensitivity of any parameter, relative to that of highest composite sensitivity, is equal to or less than this threshold, then that parameter is a candidate for sensitivity re-use on some PEST iterations.

SENRELTHRESH can be set to any non-negative value. However there is no use in setting its value above 1.0; if it is set to 1.0 or above, then the number of parameters for which sensitivity re-use is active on any iteration is set entirely by the SENMAXREUSE variable, as the relative composite sensitivity of all parameters is, by definition, less than the threshold.

SENMAXREUSE

This is the maximum number of parameters for which sensitivity re-use is activated on any one PEST iteration. If it is set to a negative number, then PEST automatically sets it to half the number of adjustable parameters. It should not be set to a value greater than the number of adjustable parameters minus 1. If it is set to zero, PEST will terminate execution with an error message.

If more parameters than SENMAXREUSE incur candidature for sensitivity reuse through possession of composite sensitivities less than or equal to the RELSENREUSE threshold, then parameters are eliminated from candidature in order of composite sensitivity, from highest to lowest, such that the total number of parameters for which sensitivities are re-used is equal to SENMAXREUSE.

SENALLCALCINT

SENALLCALCINT denotes the optimisation interval at which all parameter sensitivities will be re-calculated, regardless of their relative composite sensitivities. If, for example, this is set to 3, then all parameter sensitivities will be computed on iterations 1, 4, 7, 10 etc.

SENPREDWEIGHT

As is described in the PEST manual, the composite sensitivity of a parameter is obtained by summing elements of the column of the Jacobian matrix pertaining to that parameter, with each element in a column multiplied by the square of the observation weight pertaining to that column. If PEST is being employed in predictive analysis mode, the question then arises as to what weight the predictive sensitivity element in each column should be given. Presumably, the weight should be high enough for it to be visible in the composite sensitivity of any parameter, for getting sensitivities of the prediction right may be critical to raising or lowering that prediction as required by the predictive analysis process. Hence parameters that are critical to that prediction may not be good candidates for sensitivity reuse, for their sensitivities should be re-calculated during every optimisation iteration instead.

SENPREDWEIGHT is the weight to assign to the prediction in computation of composite parameter sensitivities when PEST is run in *predictive analysis* mode. If it is set to a negative number, PEST will compute its weight as equal to the maximum weight assigned to any other observation (including prior information if SENPIEXCLUDE is set to "no").

SENPIEXCLUDE

If a PEST control file cites prior information, then a SENPIEXCLUDE setting of "yes" will instruct PEST not to take any account of this prior information when determining composite parameter sensitivities for the purpose of deciding parameter candidates for inclusion in the sensitivity re-use process.

It should be noted. that, irrespective of any sensitivity reuse settings, regularisation observations and/or prior information equations are never included in the calculation of composite parameter sensitivities for the purpose of deciding candidature or otherwise for parameter sensitivity re-use.

2.29 Regularisation Convergence Adjustment

Under normal circumstances, when working in regularisation mode, PEST ceases execution immediately if the measurement objective function falls below its user-supplied target value

of PHIMLIM. Therefore it does not undertake further iterations in an attempt to lower the regularisation objective function any further in order to maximize the extent to which parameters adhere to the preferred condition that is encapsulated in regularisation constraints.

There are some circumstances, however, where minimization of the regularisation objective function is just as important as allowing the measurement objective function to reach PHIMLIM. In these circumstances, PEST should continue with the parameter estimation process until some other convergence criterion is met (for example that parameters cease to change noticeably between iterations, or that neither the regularisation nor measurement objective function changes noticeably between iterations). This can be achieved through setting the REGCONTINUE variable to "continue".

```
* regularisation
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE] [CONJGRAD] [CGRTOL] [CGITNLIM]
WFINIT  WFMIN  WFMAX [LINREG] [REGCONTINUE]
WFFAC  WFTOL  [IREGADJ]
```

Variables in the "regularisation" section of a PEST control file.

The above figure shows the locations of all control variables occupying the "regularisation" section of the PEST control file; optional variables are depicted in square brackets. The REGCONTINUE variable (shown in bold) is placed on the third data line of this section. Its location is interchangeable with that of the LINREG variable. In fact, if the LINREG variable is omitted from the regularisation section of a PEST control file (as it normally is), REGCONTINUE can be placed immediately following the value of the (mandatory) WFMAX variable.

If present, REGCONTINUE must be supplied as either "continue" or "nocontinue". If absent, it is assumed to be "nocontinue"; in this case PEST ceases execution as soon as the measurement objective function falls below PHIMLIM. If it is set to "continue" however, PEST will continue iterating, trying to improve the regularisation objective function, until some other convergence criterion is met.

2.30 New Termination Criteria

Two new (and optional) termination control variables named PHISTOPTHRESH and PHIABANDON have been added to the "control data" section of the PEST control file. These are located on line 9 of the PEST control file, along with other termination variables. These, and another new variable named LASTRUN, are shown highlighted in the figure below.

```
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM]
NTPLFLE NINSFLE PRECIS DPOINT NUMCOM JACFILE MESSFILE
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE]
RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND]
PHIREDSWH [NOPTSWITCH] [DOAUI]
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN] [PHIABANDON]
ICOV ICOR IEIG
```

Variables in the "control data" section of the PEST control file.

If PHISTOPTHRESH is set to a positive number, PEST will cease execution if the objective function falls below this value at the end of any optimisation iteration. (This criterion is applied to the measurement objective function if PEST is run in "regularisation" mode.) If it

is set to zero or a negative number it is ignored. Note that an error condition will occur if PHISTOPTHRESH is set to a positive number while PEST is asked to run in "predictive analysis" mode.

Another new (integer) control variable can follow PHISTOPTHRESH on the 9th line of the PEST control file; this is the LASTRUN variable. If LASTRUN is set to zero, then PEST will not undertake a final model run on the basis of optimised parameters upon termination of execution. If it is not set to zero it must be set to one (its default value).

PHIABANDON optionally follows LASTRUN. If, at the end of the first model run, and any optimisation iteration thereafter, the objective function (or measurement objective function if PEST is run in "regularisation" mode) is **above** PHIABANDON, PEST will terminate execution. Set this to a very high number, or to a non-positive number (or omit it altogether), for it to have no effect.

PHISTOPTHRESH, PHIABANDON and LASTRUN can be useful in conjunction with null space Monte Carlo analysis. Using PHISTOPTHRESH, the optimisation process can be made to cease (without the carrying out of a final model run if LASTRUN is set to 0) if the objective function is reduced to a level at which the user has deemed the model to be calibrated; PEST can then move on to its next run. On the other hand if, on undertaking the initial model run using a certain set of random parameters, the objective function is so high that it is unlikely to be lowered efficiently to a state at which the model can be considered calibrated, that PEST run can be immediately abandoned (using PHIABANDON) in the hope that the next random parameter set offers more hope.

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

If desired, a user can supply PHIABANDON values on an iteration by iteration basis. Thus, for example, a user may decide that if the objective function is lower than 10000 when calculated using initial parameters, then it is worth proceeding with the parameter estimation process. On the other hand if the objective function has not dropped to 2000 by the end of the second iteration, then the parameter estimation process should be abandoned. In this case, a filename can replace the value of the PHIABANDON variable on the 9th line of the PEST control file; PEST then reads iteration-specific PHIABANDON values from the cited "PHIABANDON schedule file", an example of which is shown below.

```
10000.0
10000.0
2000.0
```

An example PHIABANDON schedule file.

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

The following should be noted.

- If there are more entries in the PHIABANDON schedule file than the maximum

number of iterations assigned to the current parameter estimation process (i.e. NOPTMAX), PHIABANDON will ignore lines after NOPTMAX+1.

- If there are less than NOPTMAX+1 lines in the PHIABANDON schedule file, PEST will assign to missing iterations the last PHIABANDON value read from the file.
- A zero or negative PHIABANDON value (or a very large value for that matter) can be used to signify that parameter estimation abandonment is not active during the pertinent iteration.

2.31 PSLAVE Command-Line Option

If PSLAVE is started using the command:-

```
pslave /n
```

it will not cease execution upon termination of PEST execution. This can be useful in circumstances where many Parallel PEST runs must be undertaken in succession (for example in calibration-constrained Monte Carlo analysis). The runs can then be undertaken within a loop written to a batch or script file. Meanwhile, the slaves need only be started once, before the batch or script file is activated.

2.32 New Jacobian Matrix Save Option

The last line of the "control data" section of the PEST control file holds three, maybe four, variables, namely ICOV, ICOR, IEIG (and maybe the optional IRES variable). An additional variable can now be placed on this line, this being named JCOSAVEITN. JCOSAVEITN is a character variable which should be supplied as either of two character strings, these being "jcosaveitn" or "nojcosaveitn". If the former string is supplied, PEST will write a Jacobian matrix file (i.e. a "JCO file") at the end of each optimization iteration, this containing the Jacobian matrix employed for that particular iteration. The name of each such file is *case.jco.N* where *N* is the iteration number to which the JCO file pertains. Alternatively, if JCOSAVEITN is set to "nojcosaveitn" or omitted altogether, PEST will not save a progression of JCO files in this manner.

The following should be noted:

1. The JCO file containing the Jacobian matix corresponding to the iteration at which the best parameter set so far was computed, is saved to a file named *case.jco* in the usual manner, irrespective of the setting of JCOSAVEITN.
2. The JCOSAVEITN variable can be placed anywhere on the 10^{th} line of the PEST control file. However it is recommended that it be placed after the values of ICOV, ICOR, IEIG (and IRES if the latter is present).

JCOSAVEITN should be set to "jcosaveitn" if it is desired that the MULJCOSEN utility be employed for monitoring of changes to composite parameter and/or observation sensitivities as the parameter estimation process undertaken by PEST progresses.

2.33 Interim Residuals File Save Option

In addition to the JCOSAVEITN variable, the last line of the "control data" section of the PEST control file can now contain another variable, this variable having a function which is not too dissimilar from that of the JCOSAVEITN variable. The name of this variable is REISAVEITN. This is a text variable which can be set to either "reisaveitn" or "noreisaveitn"

(or omitted altogether, which is equivalent to a setting of "reisaveitn").

As is documented in the PEST manual, PEST writes an "REI file" (this being an "interim residuals file") at the end of every optimisation iteration. This records observations and corresponding model outputs computed for the best parameters achieved in the optimisation process up to that iteration. Regardless of the setting of REISAVEITN, this file is saved (and the previous one overwritten) at the end of every iteration; its name is *case.rei* where *case* is the filename base of the current PEST control file. However if REISAVEITN is set to "reisaveitn", then this same file is also saved as *case.rei.N* where *N* is the current iteration number. Thus at the end of the parameter estimation process, a sequence of files remains in the PEST working directory which collectively document the history of model-to-measurement misfit for every observation within the PEST input dataset.

The JCOSAVEITN variable can be placed anywhere on the 10^{th} line of the PEST control file. However it is recommended that it be placed after the values of ICOV, ICOR, IEIG (and IRES if the latter is present).

2.34 Group-Specific Target Measurement Objective Functions

2.34.1 General

When PEST is run in regularisation mode a value must be supplied for the PHIMLIM variable in the "regularisation" section of the PEST control file. This is the target measurement objective function. As described in the PEST manual, when running in regularisation mode the parameter estimation problem is re-formulated as a constrained minimisation problem. In this problem PEST is asked to minimise the regularisation objective function subject to the constraint that the measurement objective function rises no higher than PHIMLIM. In practice PHIMLIM may not be attainable, in which case PEST lowers the measurement objective function as far as it can. If it is obtainable, PEST ceases execution as soon as the measurement objective function falls below PHIMLIM. Alternatively, PEST can be asked to continue the inversion process until the regularisation objective function is as small as it can be while the measurement objective function is simultaneously made as close as possible to its target; this is effected through setting the REGCONTINUE variable to "continue".

In some instances it can be useful to set the target measurement objective function on an observation group by observation group basis. When this is done, PEST is asked to monitor the performance of the inversion process in lowering each group-based component of the measurement objective function to its group-specific target. If one such target it not being met, PEST raises the weights assigned to members of the pertinent observation group is that the overall objective function penalty incurred through not meeting that target is greater, thus providing PEST with more incentive to actually meet it. This, of course, provides no guarantee that the target will be met. However it does place greater emphasis on the meeting of as yet unmet targets than would otherwise be the case.

When weights are altered, so too does the current value of the measurement objective function, as indeed do target measurement objective functions. However when undertaking observation weights adjustment in the manner described above, PEST ensures that the overall target measurement objective function (which is equal to the sum of group-specific target measurement objective functions expressed in terms of original measurement weights as supplied in the PEST control file) does not change. While reporting the revised current measurement objective function to the screen and to its run record file at all stages of the

parameter estimation process, it also reports the measurement objective function, and group-specific components thereof, as calculated from original weights, both during and after the parameter estimation process. The metric by which success of the constrained optimization process is judged is that these individual measurement objective function components approach (or are less than) their group-specific measurement objective function targets (which, as is stated above, are calculated on the basis of original weights as supplied in the PEST control file). In the meantime, if the measurement objective function, as reported on all attempts to upgrade parameters using different Marquardt lambdas, is lower than the initial target (obtained through summation of individual group-specific targets) then so too will be the measurement objective function calculated using initial weights.

The following should be noted.

1. Group-specific measurement objective function targets must be supplied for all non-regularisation observation groups, or for none of them.
2. A group-specific measurement objective function target must not be supplied for a regularisation group.
3. Group-specific measurement objective function targets can only be supplied if PEST is run in "regularisation" mode.
4. If group-specific measurement objective function targets are supplied, values for PHIMLIM and PHIMACCEPT as supplied in the "regularisation" section of the PEST control file are ignored. PHIMACCEPT is internally set to 1.05 times the total measurement objective function target value.
5. If a value is supplied for FRACPHIM in the "regularisation" section of the PEST control file, this is respected. Thus if FRACPHIM times the current objective function is greater than the overall target measurement objective function, the former is accepted as the temporary target for the current optimisation iteration.
6. Internal weights re-assignment by PEST will never be such that weights for any observation group are increased by a factor of more than 128 or reduced by a factor of 128 from their original values. The increase/reduction factor per iteration is never greater than 2.0.
7. Unless IREGCONTINUE is set to "continue" PEST will cease execution when the total measurement objective function falls below the total target measurement objective function; no account is taken of whether groups-specific measurement objective functions are below their group-specific targets.

The use of group-specific measurement objective function targets cannot in themselves guarantee that these targets are all simultaneously met. However they can certainly provide some assistance to the optimization process in achieving this result. The user should try, however, to minimize the need for measurement weights adjustment by choosing original weights and group-specific targets in a manner that is considered optimal prior to commencement of the optimisation process. *For example, if it becomes apparent that a group-specific measurement objective function is very easily met, then that target should be reduced rather than being a source of unnecessary "target headroom" contributing to the total measurement objective function target.*

2.34.2 Implementation

Group-specific measurement objective function targets must be supplied in the "observation

groups" section of the PEST control file. Each such target must be placed immediately following the name of a non-regularisation observation group. If a covariance matrix is supplied for that group, the name of the pertinent covariance matrix file must follow the value of the group-specific measurement objective function target. The following figure shows an example.

```
* observation groups
obs_gp_1 45.00
obs_gp_2 67.00 covmat.dat
obs_gp_3 67.00
regul1
regul2
regul3
```

The "observation groups" section of a PEST control file, showing the use of group-specific measurement objective function targets.

2.35 Bad Derivatives Damage Mitigation - 1

2.35.1 General

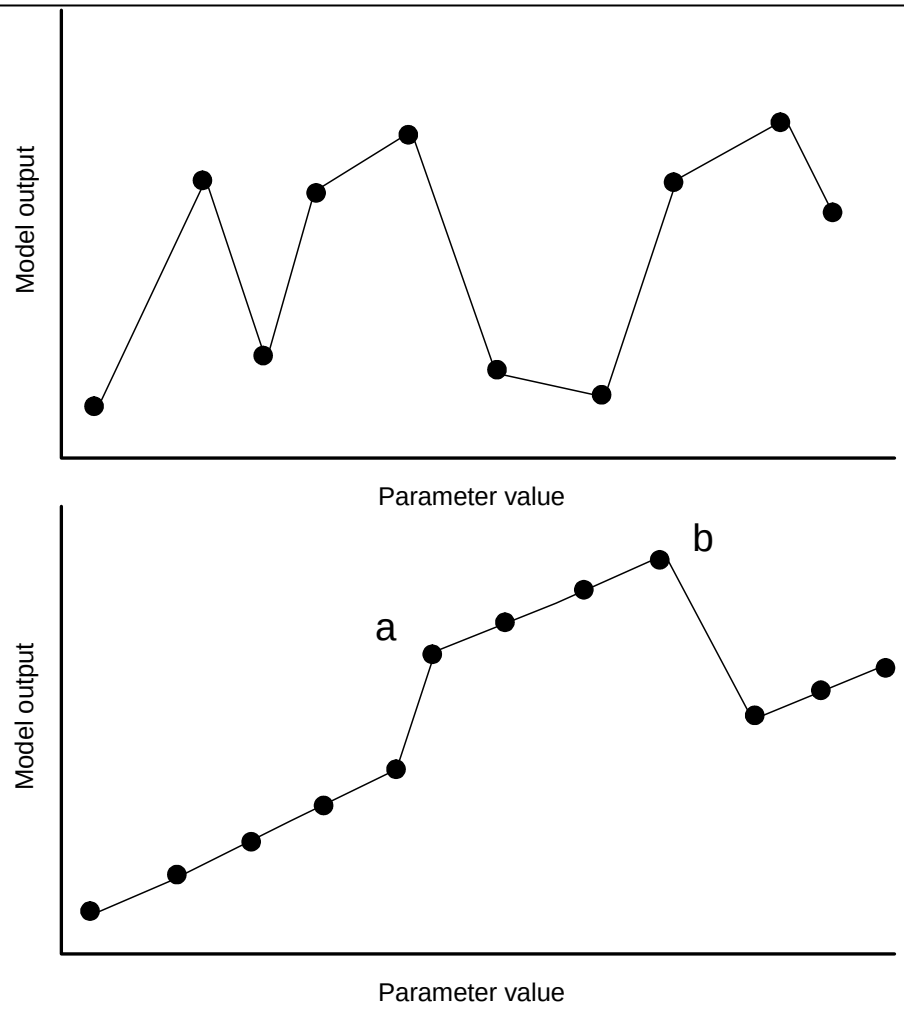
Use of the Gauss-Marquardt-Levenberg method on which PEST is based is predicated on the assumption that model outputs are differentiable (or at the very least continuous) with respect to model parameters. However often, because of poor model performance, this is not the case. Lack of differentiability can be incurred by at least the following (and many other) aspects of model behaviour:-

1. lack of convergence of iterative solvers;
2. adaptive time stepping (particularly for models which simulate groundwater/surface water interaction);
3. dry cells (if using the USGS MODFLOW groundwater model);
4. particles (if using the USGS MODPATH particle tracking model).

The JACTEST, POSTJACTEST and MULJCOSEN utilities can be used to explore the existence (or otherwise) of this problem.

Bad derivatives express themselves in various ways during a PEST run. More often than not however, a sudden refusal by PEST to lower the objective function any further (possibly accompanied by a "bouncing around" of the objective function between different Marquardt lambda) is a sign of its presence.

For many models, analysis using JACTEST reveals that contamination of derivatives by numerical noise is not necessarily purely random. The first of the following two figures illustrates a graph of one line of a JACTEST output table in which derivatives are contaminated by purely random noise; the second part however shows the more common situation, in which a plot of model outputs against incremented parameter value shows segments in which derivatives do indeed have integrity. It is only in the vicinity of points (a) and (b) in this figure that derivatives are not useable; elsewhere they are indeed fit for use in the parameter estimation process. Furthermore, it is common experience that such aberrations in slopes exist only between neighbouring pairs of points, and do not occur over parameter-space distances comprised of two or more inter-point segments.



Plots of model output vs. parameter value from a JACTEST output file.

For situations described by the second of the above figures, aberrant derivatives behaviour may be easily recognised where three-point derivatives computation is taking place. For a particular parameter, three-point derivatives computation will occur from the beginning of the parameter estimation process if FORCEN is set to "always_3" for the group to which the parameter belongs. However more commonly PEST commences the parameter estimation process with two-point derivatives for all parameters and switches to three-point derivatives for all parameters only if the parameter estimation process is proceeding very slowly. This behaviour occurs when FORCEN for all parameter groups is set to "switch".

Where three-point derivatives computation is activated, PEST has the opportunity to compare the slopes of two neighbouring segments on the curve of model output vs. parameter value. If slopes of these neighbouring segments are markedly different, this indicates that points (a) or (b) of the above graph may have been encountered. If this is the case, the derivative should be computed on the basis of only one of the neighbouring segments defined on the basis of the three parameter values, rather than on both of them. Alternatively, perhaps both segments should be abandoned and the local slope of the model output vs. parameter value curve approximated in a way that does not necessarily achieve progress of the parameter estimation process, but does not harm it either.

2.35.2 Implementation

"Spit slope analysis" is implemented according to new settings supplied in the "parameter groups" section of the PEST control file. Variables that can appear on each line of this section are illustrated in the following diagram.

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

Variables appearing in the "parameter groups" section of the PEST control file.

The SPLITTHRESH, SPLITRELDIFF and SPLITACTION variables are optional. However if one of them is supplied for a particular parameter group then all of them must be supplied for that group.

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

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

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

The following should be noted.

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

A further variable has been introduced to the "control data" section of the PEST control file which can affect the way in which split slope analysis is implemented. This is an optional variable. If present, it must immediately follow the NOPTSWITCH variable on the eighth line of the PEST control file. If present, the NOPTSWHICH variable must also be present. If

the DOAUI and/or DOSENREUSE variables are present, they must follow SPLITSWH. This is illustrated in the control data specification list below.

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

Specifications for "control data" section of PEST control file.

If SPLITSWH is provided with a zero or negative value, it is ignored. If not, it can be used to determine the point within a PEST run at which operation of split slope analysis begins. Suppose that this is set to 1.10. Then split slope analysis will begin when the following conditions have been met.

1. Central derivatives are computed for at least one parameter group. (This can be delayed by setting FORCEN for respective parameter groups to "switch").
2. If switching in this manner occurs, at least one iteration has elapsed since this switch has taken place. (This gives central derivatives without split slope analysis a chance to lower the objective function.)
3. The ratio of the best objective function achieved during a certain iteration to that achieved in the previous iteration is 1.10.

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

At the time of writing, use of split slope analysis appears to be moderately successful in allowing the parameter estimation process to progress, even where derivatives are of poor quality. Suitable settings for SPLITTHRESH, SPLITRELDIFF and SPLITACTION are still being acquired from experience. The JACWRIT utility can be employed to learn of the expected values of model output derivatives with respect to various parameters to help in the setting of these variables. Values of 1.0E-4, 0.5 and "smaller" appear to work well. However the author would welcome suggestions from user experience.

2.36 Bad Derivatives Damage Mitigation - 2

2.36.1 General

Corrupted derivatives are often recognised by the fact that their values are unusually high. However it is very difficult to know how high these values must be to qualify as being such. Another feature of corrupted derivatives is that if variation of one particular parameter causes a certain aspect of model functionality to cross an internal threshold that causes a discontinuity to arise in the relationship between that parameter and model outputs, then an entire column of the Jacobian will thus be corrupted because all model outputs will be affected. It follows that if this column is removed from the parameter estimation process for the current iteration, progress in lowering the objective function can still be made as the presence of this corrupted column will then not impede the ability of other parameters to alter their values in ways that lower the objective function.

One possible manner in which mitigation of bad derivatives may then proceed during any particular iteration of the parameter estimation process is as follows:-

1. Attempt a parameter upgrade.
2. If the objective function does not improve, identify the parameter with maximum composite sensitivity and remove that parameter from the upgrade calculation process.
3. Attempt another parameter upgrade.
4. Repeat steps 2 and 3 until either the objective function is lowered, or until it becomes obvious that it will not be lowered through a continuation of this process.
5. During the next optimisation iteration (when a fresh Jacobian matrix is available) repeat this process.

This sequence of steps is very similar to that employed by PEST's "automatic user intervention" functionality and, in fact, can be implemented with very few alterations to that algorithm. The major difference is that instead of temporarily removing parameters from the parameter upgrade process in order of increasing composite sensitivity (starting from that of lowest composite sensitivity), parameters are removed in order of decreasing composite sensitivity, starting from that of highest composite sensitivity. (Hence it is important that PEST be presented with a well-posed inverse problem for which traditional automatic user intervention is not required.)

2.36.1 Implementation

The above process is most easily implemented simply by setting the DOAUI variable in the "control data" section of the PEST control file to "auid". However this is not allowed if the SVD or LSQR solution schemes are invoked, because present PEST functionality is such that automatic user intervention is disallowed under these circumstances. Note, however, that automatic user intervention can presently be used successfully if PEST is undertaking SVD-assisted parameter estimation.

If no "automatic user intervention" setting is present in the PEST control file, control variables that affect the operation of the automatic user intervention process when used in this manner for mitigation of problems caused by bad derivatives are assigned the values shown in the following table.

Variable name	**Default Value**
MAXAUI	3*NESPAR/4
AUISTARTOPT	1
NOAUIPHIRAT	0.9
AUIRESTITN	0
AUISENSRAT	1
AUIHOLDMAXCHG	0

AUINUMFREE	1
AUIPHIRATSUF	0.8
AUIPHIRATACCEPT	0.99
NAUINOACCEPT	3*MAXAUI/4

Default values used for automatic user intervention control variables when the DOAUI variable is set to "auid". NESPAR in the above table is the number of adjustable parameters.

Note that the AUIPHIRATSUF variable takes on a slightly different role when DOAUI is set to "auid" from that which it assumes when DOAUI is set to "aui". In the latter case it refers to the ratio of the current objective function to that achieved on the current iteration with no parameters temporarily frozen; in the former case it is the ratio of the current objective function to the that prevailing at the commencement of the current iteration.

Both normal and parallel PEST can employ automatic user intervention. In the latter case the efficiency of the process is greatly increased if the PARLAM variable in the PEST run management file is set to a value that allows parallelisation of the Marquardt lambda search.

If you feel that PEST is trying to do too much automatic user intervention during any particular optimisation iteration, the easiest way to reduce the time spent on progressively freezing parameters and re-attempting upgrade calculations is to place an "automatic user intervention" section in the PEST control file, assigning the above default values to all variables contained therein except for MAXAUI which can be set to an appropriately low number. PEST will then limit its progressive removal of Jacobian matrix columns to MAXAUI before abandoning automatic user intervention for that optimisation iteration and moving on to the next iteration.

2.37 Kullback-Leibler (K-L) Information Loss Statistics

PEST now computes AIC, AICc, BIC and KIC statistics. These are recorded on its run record file, just before the parameter covariance matrix. Formulae used by PEST are as follows. (These are the same as those employed by Poeter and Hill, 2007).

$$\text{AIC} = n\ln\left(\sigma^2\right) + 2k$$

$$\text{AICc} = n\ln\left(\sigma^2\right) + 2k + \left(\frac{2k(k+1)}{n-k-1}\right)$$

$$\text{BIC} = n\ln\left(\sigma^2\right) + k\ln(n)$$

$$\text{KIC} = (n-(k-1))\ln\left(\sigma^2\right) - (k-1)\ln(2\pi) + \ln\left|\mathbf{X}^t\mathbf{Q}\mathbf{X}\right|$$

where:

n is the number of non-zero-weighted observations and prior information equations employed in the parameter estimation process;

k is equal to *m*+1 where *m* is the number of adjustable parameters involved in the parameter estimation process;

σ^2 is the residual variance, calculated as Φ_{min}/n, where Φ_{min} is the minimized

objective function;

X is the Jacobian matrix computed on the basis of optimised (or almost optimised) parameters;

Q is the weight matrix employed by PEST.

The following should be noted.

1. The above statistics are calculated only if PEST is run in "estimation" mode; they are not calculated if PEST is run in "predictive analysis" or "regularisation" modes. Nor are they computed if solution of the inverse problem is achieved using truncated singular value decomposition or LSQR.
2. If a problem is so ill-determined that the covariance matrix cannot be calculated (by virtue of the fact that $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ cannot be inverted), then these statistics will not be calculated either.

The author must confess that in contexts where highly parameterised inversion and concomitant uncertainty analysis can be carried out using efficient, state-of-the-art regularisation methodologies described herein, he can see little use for these statistics, for they ignore the fact that the world is a complex place and that "maximum likelihood" cannot be computed without recognition of this complexity (as encapsulated in the C(**p**) matrix used elsewhere in this document). Statistics such as those computed by the PREDVAR1 utility described herein recognise this complexity; they also recognise the fact that regularisation of one kind or another is required for unique solution of an inverse problem and that the "cost of uniqueness" is a significant loss of resolution in representing hydraulic properties in a calibrated model. The PREDVAR methodology thus provides a more coherent approach to estimation of the optimum number of parameters to employ in solving an inverse problem, as loss of system detail incurred by using too few parameters is played against amplification of observation noise incurred by using too many. Furthermore, methodologies such as Null Space Monte Carlo allow a modeller to avoid having to commit him/herself to a unique set of parameters at all, and to analyse the uncertainty of model predictions with full account taken of the fact that the calibration process can normally capture very little of the true complexity of real-world systems. The need for "model averaging" techniques, especially in contexts where the very existence of discrete models whose relative worth must be analysed is an artificial outcome of an inability on the part of older-style parameter estimation software to accommodate complex but continuous parameter fields, is highly questionable in these circumstances.

2.38 Subspace-Enhanced Tikhonov Regularisation

2.38.1 General

Use of Tikhonov regularisation in the inversion process often results in parameter fields that "look good" from a geological or other perspective. This will be especially the case if Tikhonov constraints have been formulated in such a way that the "default parameter condition" that they express represents an expert's encapsulation of pre-calibration parameterisation maximum likelihood. However not only should the preferred condition express parameterisation reasonableness; in addition to this, weights and/or covariance matrices applied to regularisation constraints should be such that violation of these constraints in order to allow model outputs to fit field data takes place in a reasonable, or "least-unlikelihood" fashion. In the groundwater modelling context, this will result in the

introduction of heterogeneity that is of a scale and disposition that "makes sense" in the geological setting under consideration.

A practical problem that often arises when using Tikhonov regularisation is that neither the "default condition" of real-world parameters, nor the covariance matrix of spatial or temporal parameter heterogeneity that describes their potential variability, is known (or even very applicable). Another problem is that, as the optimisation process advances and PEST pursues a good fit with field data in areas of high spatial data density where there may be strong evidence for the existence of parameterisation heterogeneity, it "lets go" of Tikhonov regularisation constraints in order to achieve these fits, and to therefore express the heterogeneity which they suggest. "Letting go" takes place through reduction of the regularisation weight factor. However an unfortunate consequence of "letting go" is that numerical instability of the inversion process may be incurred, for the consequential lack of constraints on parameters in data-poor parts of the model domain may result in nonuniqueness in estimation of parameters in those areas. Condition numbers then rise, and the consequential numerical instability will almost certainly hamper further progress of the parameter estimation process.

This problem can be partially overcome through applying differential weighting to regularisation constraints, with weights being weaker on parameters (and/or parameter combinations) that are relatively estimable on the basis of the current calibration dataset, while being stronger on constraints that pertain to parameters (and/or parameter combinations) that are relatively inestimable. Thus the information content of the calibration dataset can be transferred to parameters (and parameter combinations) of which this dataset is informative, without the need to relax application of default conditions on parameters (and parameter combinations) for which information in the calibration dataset is weak or absent.

PEST is given license to adopt differential weighting for regularisation prior information equations and regularisation observations through use of the IREGADJ regularisation control variable. Differential weighting is computed and applied on a group-by-group basis if IREGADJ is set 1, 2 or 3. If it is set to 4 or 5, regularisation weights adjustment takes place on an item-by-item basis, where an "item" is an individual regularisation prior information equation, or an individual regularisation observation. These latter options will now be described.

2.38.2 Solution Space Projection

Let **X** represent the Jacobian matrix for the current parameter estimation problem, and let **p** represent parameters that are featured in this problem. Let **Q** represent the observation weight matrix. Let singular value decomposition of the weighted Jacobian matrix lead to the following equation:

$$\mathbf{Q}^{1/2}\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^{t} \qquad (2.38.1)$$

where, as usual, the columns of **V** comprise orthogonal unit vectors which collectively span parameter space. On the basis of a user-selected singular value threshold, **V** can be subdivided into submatrices $\mathbf{V}_1$ and $\mathbf{V}_2$ such that:

$$\mathbf{Q}^{1/2}\mathbf{X} = \mathbf{U}\begin{bmatrix}\mathbf{S}_1 & \mathbf{S}_2\end{bmatrix}\begin{bmatrix}\mathbf{V}_1^{t} \\ \mathbf{V}_2^{t}\end{bmatrix} \qquad (2.38.2)$$

Often the threshold will be chosen such that $\mathbf{V}_1$ spans the "calibration solution space" (combinations of parameters to which model outputs corresponding to observations

comprising the calibration dataset are sensitive), while $\mathbf{V}_2$ spans its orthogonal complement, the "calibration null space".

Let the vector **y** represent the sensitivities of a regularisation prior information equation, or regularisation observation, to the parameters **p**. If **y** has a large projection onto the calibration solution space, then the model-generated outcome of the prior information equation or observation is in fact well determined on the basis of information gained through the calibration process. However if the "observed" value of this equation or observation contradicts its value as generated by the calibrated model, then PEST will reduce the weight applied to this particular regularisation constraint (and hence to all regularisation constraints if IREGADJ is provided with a value of zero). The situation described above whereby progress of the parameter estimation process risks being hampered by numerical instability can then arise.

On the other hand, if the vector **y** has a small projection onto the calibration solution space, then the calibration dataset provides no information that contradicts the user-supplied value of this constraint, irrespective of its value. Hence this constraint can (and should) be tightly applied, with its weight maintained or increased in order to ensure that this occurs.

2.38.3 IREGADJ Settings

Subspace enhancement of Tikhonov regularisation is implemented by PEST in one of two ways, depending on whether the IREGADJ variable is set to 4 or 5. If IREGADJ is set to 4, PEST follows the procedure set out below in assigning weights to individual regularisation observations and prior information equations.

1. PEST formulates $\mathbf{Q}^{1/2}\mathbf{X}$ on the basis of the current Jacobian matrix **X**, but from which all regularisation constraints are omitted. Thus **X** features only the calibration dataset.
2. Singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$ is undertaken – see equation (2.38.1).
3. From the current Jacobian matrix PEST obtains the sensitivity vector **y** pertaining to each regularisation observation and each regularisation prior information equation included in the current calibration process.
4. For each such **y**, for example $\mathbf{y}_j$, the direction cosine:

$$\cos(\theta_{ji}) = \frac{\mathbf{y}_j^t \mathbf{v}_i}{\sqrt{\mathbf{y}_j^t \mathbf{y}_j}} \qquad (2.38.3)$$

   is computed for each column $\mathbf{v}_i$ of **V**.
5. The weight for regularisation constraint *j* is calculated as:

$$w_j = \sum_i \frac{\cos(\theta_{ji})}{s_i} \qquad (2.38.4)$$

   where s_i is the singular value associated with the unit vector $\mathbf{v}_i$.

This strategy results in small weights being assigned to regularisation constraints that are informed by the data (and therefore run the risk of contradicting that data) and large weights being assigned to regularisation constraints that are not informed by the data, and hence which are needed for stable estimation of the parameters that they cite. However application of (2.38.4) can result in weights for regularisation constraints which can vary over many orders of magnitude. Two devices are employed to limit this variability.

The first of these devices requires no input by the user, as it is implemented automatically by PEST. In calculating w_j through (2.38.4), PEST will not employ an s_i value which is less than 10^{-7} of the maximum singular value pertaining to the current $\mathbf{Q}^{1/2}\mathbf{X}$ matrix. Instead, a value of 10^{-7} times the maximum singular value is employed in place of singular values which are less than this.

Secondly, once weights for all regularisation constraints have been computed using (2.38.4), the range of regularisation weight magnitudes is compressed such that the ratio of the largest to the smallest resulting weight is equal to a user-specified value. Weights compression can be linear or logarithmic. In the former case, alterations to weight magnitudes are proportional to those magnitudes as calculated using (2.38.4). In the latter case, alteration is proportional to the logs of weight magnitudes.

If IREGADJ is set to 5 rather than 4, each regularisation observation and prior information equation is assigned either of two weight values, with the ratio of these values being set by the user. (The actual values are adjusted during each optimisation iteration in accordance with PEST's normal computation of the regularisation weight factor; however relativity between them is maintained.) The procedure is as follows.

1. The user selects a singular value ratio for which $\mathbf{V}$ of equation (2.38.1) is partitioned into $\mathbf{V}_1$ and $\mathbf{V}_2$ of equation (2.38.2).
2. If, for regularisation constraint *j*, the sum of squared direction cosines $\cos(\theta_{ji})$ onto eigenvectors comprising the columns of $\mathbf{V}_1$ is greater than 0.5, the regularisation constraint is assigned the lower of the two user-supplied weights.
3. If the sum of squared direction cosines $\cos(\theta_{ji})$ onto eigenvectors comprising the columns of $\mathbf{V}_1$ is less than 0.5, the regularisation constraint is assigned the greater of the two user-supplied weights.

It must be remembered that, irrespective of whether IREGADJ is set to 4 or 5, regularisation weights are subjected to global weights adjustment during each optimisation iteration in accordance with PEST's normal procedure for implementing Tikhonov regularisation. However in accordance with PEST's weights adjustment procedure, relativity between the weighting assigned to regularisation observations and prior information equations is preserved.

2.38.4 Implementation of Subspace-Enhanced Tikhonov Regularisation

Three new variables are introduced to the "regularisation" section of the PEST control file to govern implementation of subspace-enhanced Tikhonov regularisation. The figure below depicts this section of the PEST control file.

```
* regularisation
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE]
WFINIT  WFMIN  WFMAX  [LINREG] [REGCONTINUE]
WFFAC  WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]
```

If IREGADJ is set to 4, values must be supplied for NOPTREGADJ and REGWEIGHTRAT; if it is set to 5, a value must also be supplied for REGSINGTHRESH.

NOPTREGADJ (an integer) specifies the optimisation interval at which regularisation weights are re-calculated in the manner discussed above. If the number of observations and/or parameters involved in the current parameter estimation process is large, undertaking singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix may be a computationally expensive exercise. The user may thus wish to avoid having to do this during every optimisation iteration. This can be

achieved by setting NOPTREGADJ to a number greater than 1. Thus if, for example, NOPTREGADJ is set to 3, regularisation weights re-calculation will take place during every 3rd iteration. Note that it is always undertaken on the first iteration of the parameter estimation process.

REGWEIGHRAT (a real number) specifies the ratio of the highest to lowest regularisation weight enforced through the weights compression process described above. A setting of between 10 and 100 is suggested; however this suggestion may change as further experience in the use of subspace-enhanced Tikhonov regularisation is gained.

The REGSINGTHRESH (real) variable is required in the PEST control file only if NOPTREGADJ is set to 5. It defines the singular value ratio which is deemed to separate the calibration solution space from the calibration null space for the purpose of assigning a weight to each regularisation observation and prior information equation in the manner described above. For consistency with the EIGTHRESH variable used in the "singular value decomposition" section of the PEST control file, this ratio actually applies to singular values of the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix rather than those of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix (the former are squares of the latter). This being the case, a lower limit of about 10^{-7} is suggested for this variable, this corresponding approximately to a suitable level for separation of the solution space from the null space where there is little or no measurement noise associated with the calibration dataset. More commonly, however, this should be set to a much higher value, for example 10^{-5}. However, as for other suggested values for the control variables discussed herein, this suggestion may be modified with experience.

If IREGADJ is set to 5, the REGWEIGHTRAT variable determines the ratio of null space to solution space regularisation weights. NOPTREGADJ determines the iteration interval at which regularisation weights adjustment takes place.

2.39 Forgiving Failed Model Runs

Sometimes, when a model is provided with a particular set of parameters, it fails to run to completion. Its output file is not then written. PEST reports this error condition and ceases execution. This situation is most likely to arise during those phases of the parameter estimation process where PEST is altering the value of the Marquardt lambda and testing new upgraded parameter values; this is the stage of the parameter estimation process where parameters are most likely to be different from their predecessors, and hence where model run failure is most likely to occur. Under these circumstances, it is better to attempt calculation of other parameter upgrades that do not instigate model failure than to abort the parameter estimation process entirely. The new LAMFORGIVE variable allows this to happen.

LAMFORGIVE should be placed at the end of the sixth line of the PEST control file. This is the same line as that on which other variables pertaining to use of the Marquardt lambda are placed. It should be placed after the NUMLAM variable. It can be the sixth or seventh variable on the line. If an optional JACUPDATE variable is placed after NUMLAM, LAMFORGIVE can either precede it, or it can be placed after it.

LAMFORGIVE must be supplied as "lamforgive" or "nolamforgive". It it is omitted, the default value of "nolamforgive" is used.

If LAMFORGIVE is supplied as "lamforgive", then PEST will not cease execution on model run failure, if that failure occurs during the Marquardt lambda search procedure. Instead, it will try a number of different lambda values. Lambda values which it uses in this process are

calculated from previous ones using the same procedure as for normal lambda selection. (The value supplied for LAMFAC is most pertinent in this regard.) However if after a few attempts (normally four), it is found that no lambda value leads to an upgraded parameter set that allows the model to complete execution, PEST takes appropriate action. It either switches to central derivatives calculation, or ceases execution altogether (with an appropriate error message).

2.40 Slave Groups

Parallel PEST optionally allows slaves to be gathered into groups. If this is done, the name of the group to which each slave is assigned must follow the name of its working directory in the Parallel PEST run management file. The figure below shows an example.

```
Prf
3  0   0.10000    -2 0
Pest_slave_1  .\test1   "group1"
Pest_slave_2  .\test2   "group1"
Pest_slave_3  .\test3   "group2"
1.0 1.0 1.0 1.0
```

Example of a Parallel PEST run management file showing slave groups.

The following rules must be followed when assigned slaves to groups.

1. If any slave is allocated to a group, all slaves must be allocated to a group.
2. If a group name contains a space, this name must be surrounded by quotes when supplied in the run management file. If not, quotes are optional.
3. Slave group names are case-insensitive; these names are converted to lower case internally.

The group concept is used in allocating runs to slaves. When undertaking model runs on the basis of different Marquardt lambdas, the number of runs that must be undertaken through any run packet may be considerably less than the number of slaves at PEST's disposal. In this case, PEST attempts to allocate runs preferentially to slaves belonging to different groups; it will not allocate two model runs to a particular slave group as long as all slaves belonging to another group are idle. Thus, for example, where slaves are designed to use different nodes on a relatively small number of different machines, and are subdivided into groups on the basis of the machine to which each node belongs, this strategy minimizes the run load on any single machine. Hence if slaves are allocated to machine-specific groups, then if a machine is available on which no model run is taking place, this will be allocated a model run instead of a machine on which a run is already taking place.

2.41 Five Point Derivatives Stencil

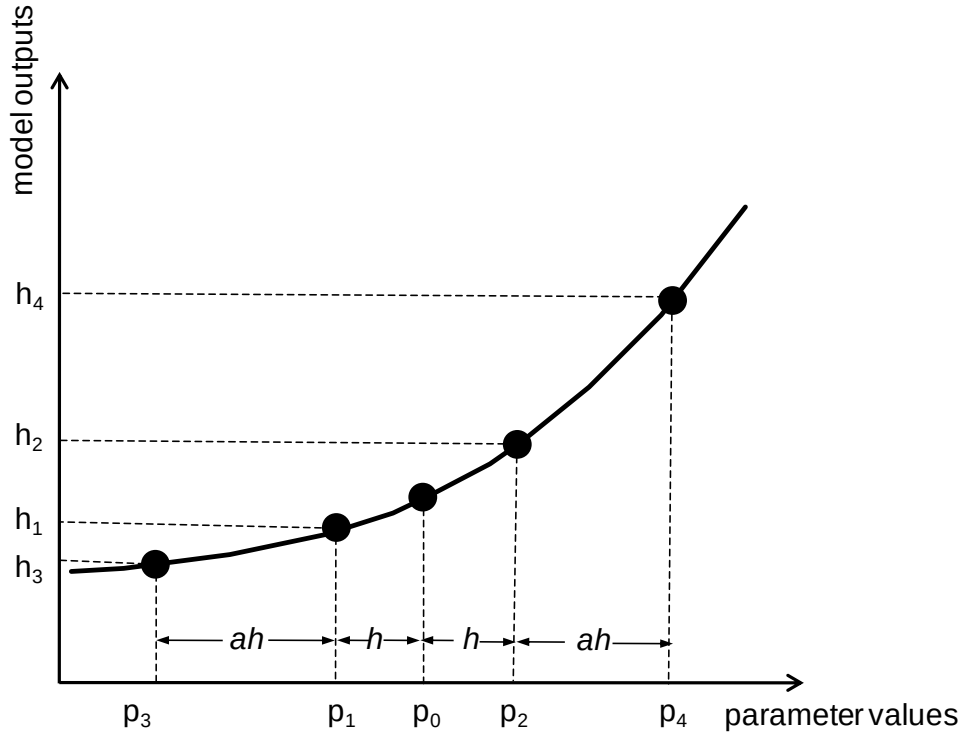
2.41.1 General

When computing upgrades for parameters, much hinges on the integrity of the Jacobian matrix. PEST normally fills this matrix itself, calculating derivatives using finite parameter differences. Differences can be calculated on the basis of two parameter values, or using three parameter values. In the former case, one run per parameter is required for filling of the Jacobian matrix during each iteration of the parameter estimation process. In the latter case, two runs are required per parameter per iteration.

A new derivatives calculation option allows the use of four parameter values in calculation of

numerical derivatives. Parameter values used in this procedure are symmetrically disposed with respect to the current parameter value. Four model runs per iteration are required when derivatives are calculated in this way. In spite of the fact that only four parameter values are required, the scheme is referred to as a "five point stencil" because the derivative that is calculated in this way pertains to the central (unused) point corresponding to the current value of the parameter.

The situation is depicted in the following figure.



Five point derivatives stencil.

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

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

James Nearing, in "*Mathematical Tools for Physics*" downloadable from:

http://www.physics.miami.edu/~nearing/mathmethods/

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

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

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

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

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

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

The latter choice is far better where model outputs are contaminated by numerical noise due

to model solution convergence difficulties and other factors. It would normally be to address this problem that a modeller would even consider undertaking four runs per parameter per iteration.

2.41.2 Implementation

PEST control variables which govern the calculation of derivatives are located in the "parameter groups" section of the PEST control file. These variables are supplied on a group-by-group basis.

To activate the five point stencil, set FORCEN to "switch_5" or "always_5". If this is done DERMTHD must be set to "minvar" to activate the minimum error variance alternative (which is recommended), or "maxprec" to activate the maximum precision alternative. The distance 2*h* in the above figure is set to the normal parameter derivative increment (calculated according to DERINC and DERINCLB) multiplied by the central derivatives increment expansion factor DERINCMUL. The factor *a* is set to 3.0; hence the increments p_4 to p_1, p_1 to p_2 and p_2 to p_4 are identical.

The following should be noted:

1. If FORCEN is set to "switch_5", the switch from two point derivatives to a five point stencil takes place according to the same criteria as would be used to switch to three point derivatives if FORCEN were set to "switch"; this is governed by the settings of the PHIREDSWH and NOPTSWITCH variables.
2. Bad derivatives mitigation through split slope analysis is disabled if a five point derivatives stencil is employed.

2.42 Parallel Run Queue

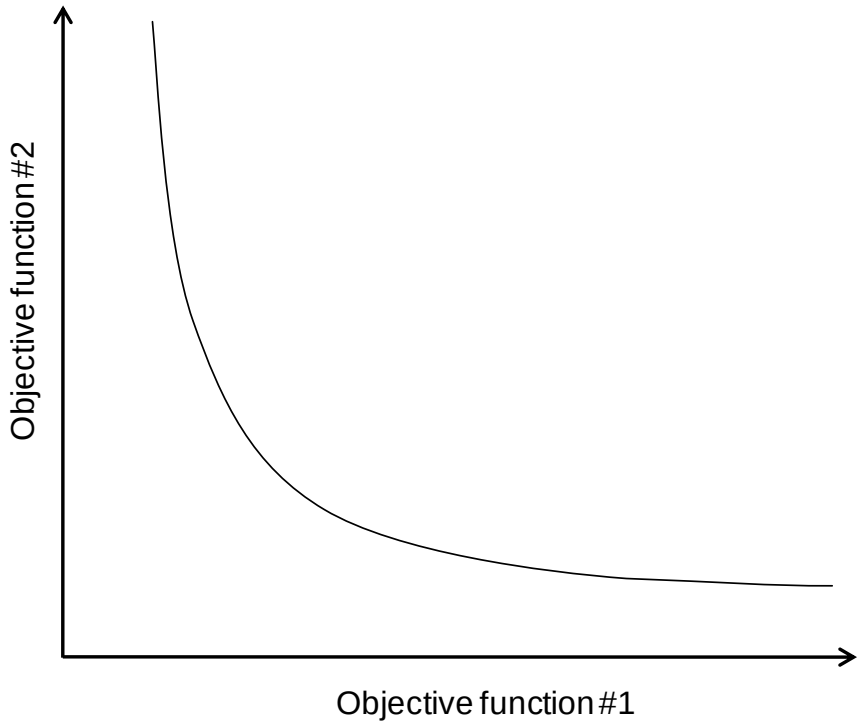
Parameter values corresponding to queued parallel runs, and model output values calculated on the basis of completed parallelized runs, are no longer stored internally by PEST. Instead they are now stored in direct access files; where parameter and observation numbers are large, this strategy can result in far smaller memory usage by PEST than would otherwise be the case. These direct access files are named *pest###.dap* (for parameter values) and *pest###.dao* (for model output values). These are normally deleted at the end of a PEST run. However if PEST is stopped prematurely (for example using the Ctl-C keys) they may still be present. Feel free to delete these files yourself as PEST will delete them anyway at the commencement of its next run in that same working directory.

2.43 Pareto Mode

2.43.1 General

The discussion in this section complements that provided in Section 4.25 of this document, where the use of PEST's "Pareto" mode in exploration of model predictive uncertainty is presented

When an objective function possesses two components it may not be possible to minimize both components simultaneously because minimization of one may compromise minimization of the other, and vice versa. Under these circumstances it is possible to define a "tradeoff curve" between the two objective functions by varying parameters appropriately; see the figure below.



A Pareto front defining the tradeoff between two objective functions.

The "Pareto front" is defined as the locus of points in parameter space along which it is not possible to lower both objective functions simultaneously. At one end of the Pareto front, one of the two objective functions is minimized, while at the other end of the Pareto front the other objective function is minimized. (Note that a Pareto front can exist between more than two objective function components, in which case its definition is multi-dimensional rather than two-dimensional as in the above case. However only two components are considered in the present discussion, and in the design of PEST's Pareto front exploration capabilities.)

When run in "Pareto" mode, PEST's task is to start at one end of the Pareto front and work its way along the front towards its other end. In most cases of practical interest, there is no need for PEST to actually reach the other end of this front, for much can often be learned through traversing just a part of it.

Settings in the PEST control file through which PEST is instructed to explore part of the Pareto front are provided in Section 4.25 of this document. Also discussed in that section is the information that PEST provides as it traverses the front. This information is contained within two files, these being named the "Pareto objective function file" and the "Pareto parameter data file". The first, an ASCII file, is named *case.pod* where *case* is the filename base of the PEST control file; the second, a binary file, is named *case.ppd*. The contents of this latter file can be examined using the PPD2ASC and PPD2PAR utilities described in Section 4.25. In addition to this, PEST writes multiple parameter values files; the parameter value file which contains parameter values achieved at the end of optimization iteration *N* is named *case.par.N*.

When PEST is run in "Pareto" mode, the PESTMODE variable in the "control data" section of the PEST control file must be set to "Pareto" rather than to "estimation", "predictive analysis" or "regularization". A "Pareto" section must also be present at the end of the control file. The contents of this section are described in detail in Section 4.25. The discussion of the present section is restricted to use of PEST's Pareto functionality in achieving sensible

parameter fields in highly-parameterized inversion.

2.43.2 Pareto Concepts in Highly Parameterized Inversion

When PEST is run in "regularization" mode, two mega-components comprise the overall objective function. One component is named the "measurement objective function" while the other component is named the "regularization objective function". PEST adjusts weights on observations (including prior information equations) comprising the regularization objective function such that the measurement objective function approaches (if possible) a user-supplied target value. If observations and prior information equations that collectively define regularization constraints are pervasive (and adequately reflect knowledge of site conditions), and if the target measurement objective function is set at a value that reflects the level of noise associated with the measurement dataset, the calibrated parameter field will deviate from a presumably maximum-likelihood condition described by regularization constraints only to the minimum amount required to achieve the desired level of fit.

In practice, selection of an appropriate value for the target measurement objective function is difficult, for the amount of noise associated with the measurement dataset is unknown. Furthermore, much of this "measurement noise" is actually structural noise which, as is the nature of structural noise, possesses an unknown (though probably singular) covariance matrix. Model-to-measurement fit is normally construed to be "too good" when the cost of achieving this fit is the introduction of too much heterogeneity to the calibrated parameter field. When this occurs (as it often does), the regularized inversion process is then repeated with the target measurement objective function set higher, for it can be concluded from the previous calibration exercise that there is more "noise" in the measurement dataset than was previously thought. Eventually a suitable compromise is found between goodness of fit on the one hand, and acceptability of the calibrated parameter field on the other hand. The model is then deemed to be "calibrated".

From the above analysis it is obvious that there is an interplay between goodness of fit and calibrated parameter field acceptability. In practice, a suitable level of model-to-measurement fit is not known in advance of the calibration process because the level of structural noise associated with model outputs corresponding to field measurements is also unknown in advance. In fact the level of structural noise can only be assessed through the "damage" done to parameter fields in achieving various levels of fit. This "damage" can, in turn, only be assessed through the calibration process. Hence part of the role of the calibration process is to seek a suitable compromise between model-to-measurement misfit and parameter field acceptability. In doing this, the calibration process informs the user of at least some of the stochastic characteristics of structural noise. The source of this information is ultimately the user's concept of the level of heterogeneity that realistically prevails in system properties that parameters purport to represent in the model that is currently being calibrated.

Pareto concepts can be used to assist in this necessarily subjective process of reconciling goodness of fit with parameter field heterogeneity as it is undertaken through the model calibration process. In implementing this methodology the modeler begins with a parameter field for which the regularization objective function is zero. (Note however that this objective function component is not referred to as such when PEST runs in "Pareto" mode – see below.) Such a parameter field is normally easy to obtain. In many cases it will be piecewise uniform; in other cases, parameter values will simply be their preferred values as suggested by site characterization studies. However the user should endeavour to supply weights to prior information equations and regularization observations that reflect the expected level of

hydraulic property variability that prevails at a particular study site. In particular, weights should be roughly equal to the inverse of the standard deviation of expected parameter variability. Alternatively, observations and prior information equations which provide the "soft knowledge" that often forms regularization constraints should be supplied a covariance matrices based on one or a number of variograms that are presumed to characterize parameter field variability (see, for example, the PPCOV program from the PEST Groundwater Data Utilities). Regardless of how weighting is assigned however, it is particularly important to establish correct relativity between parameters of different types. Once weights have been assigned, the regularization component of the calibration dataset, taken on its own, should provide a pre-calibration description of parameter site variability as far as this is known by the user – this descrption including preferred parameter values (and/or parameter relationships), together with a stochastic characterization of the variability of these values/relationships.

If a weight multiplier of zero is applied to all observations comprising the measurement objective function (it is not referred to as such when PEST runs in "Pareto" mode – see below), then not only is the regularization objective function equal to zero, for so too is the measurement objective function. Use of such a weight multiplier is most easily attained by running PEST in "Pareto" mode, assigning all observations to a single observation group, citing this group as the value of the PARETO_OBSGROUP Pareto control variable, and setting PARETO_WTFAC_START to zero.

As for regularization observations and prior information equations, care should be taken to ensure correct relativity of weighting between different measurement types when assigning them to this over-arching observation group. In some cases weighting should be inversely proportional to expected measurement credibility. In other cases, structural aspects of model-to-measurement misfit are best handled by appropriate processing of model outputs and corresponding observations, assigning processed outputs to different groups (before combining them into a single group as discussed above), and ensuring that each such group contributes roughly the same to the starting objective function (using, for example, the PWTADJ1 utility). Also, if possible, measurement weighting should be such that the yet-to-be-determined optimized measurement objective function is of the same order of magnitude as the optimized regularization objective function when the weight factor for the former about 1.0. This increases the chances that a single traversal of the Pareto front with PARETO_WTFAC_FIN set to between 1.0 and 2.0 will locate a point in parameter space that the user considers to be optimal.

PEST can now be run in "Pareto" mode; as it does so, measurements comprising the observation dataset are introduced slowly to the overall objective function as PEST increases the weight multiplier that is applied to this overall group. The PARETO_WTFAC_START control variable should be set above zero; better still, it could be set to zero and the NUM_ITER_START variable set to zero. Either of these measures ensures that PEST's first iteration is not devoted to lowering an initial objective function that is already zero (which, of course, is a futile undertaking); the latter option however adds the zero objective function to the *pod* file which can be useful for plotting purposes. Meanwhile, PARETO_WTFAC_FIN should be set at a level that hopefully achieves slightly too good a model-to-measurement fit. A setting of between 1.0 and 2.0 will presumably achieve this if the above weighting rules are followed. Traversal of that portion of the Pareto front so defined will then provide parameter sets from which the user can choose one that he/she considers to provide optimal tradeoff between parameter field credibility and goodness of model-to-measurement fit.

2.43.3 Implementation Details

In undertaking regularization through traversal of the Pareto front, the following steps should be taken. (Some of which have already been discussed.)

1. All observations comprising the measurement dataset should be assigned to a single observation group. To the best of the user's ability, relative weighting assigned to objective function components within this overall group should be correct. This may have been achieved by equalizing contributions to the total objective function by different component observation groups before combining them into a single group. Also, weights assigned to members of this group should be such that, as far as a user can judge, the contribution to the overall objective function made by regularization observations and prior information equations on the one hand, and by measurements on the other hand, will be about equal at that point on the Pareto front that is considered to be optimal.
2. Parameters should be given initial values which result in a zero value for the component of the objective function contributed by regularization observations and prior information equations. Meanwhile weights assigned to these entities should be the inverse of the expected standard deviation of variability of parameters of different types (or of relationships between them enshrined in these equations) as discussed above.
3. PESTMODE should be set to "Pareto".
4. A "Pareto" section should be added to the PEST control file. The name of the observation group holding all measurement observations should be assigned to the PARETO_OBSGROUP variable. With the weighting strategy suggested above, settings for other Pareto control variables should be as follows.
    a. 0.0 for PARETO_WTFAC_START
    b. 1.5 to 2.0 for PARETO_WTFAC_FIN
    c. 15 to 20 for NUM_WTFAC_INC
    d. 0 for NUM_ITER_START
    e. 2 for NUM_ITER_GEN
    f. 2 for NUM_ITER_FIN
5. Run PEST.

It is important to note that observations and prior information equations that comprise regularization constraints do not need to belong to an observation group whose name begins with "regul", as must be the case when PEST is run in "regularization" mode. However, it is probably a good idea to maintain this protocol for ease of identification of these entities. Also, PEST does not refer to a "measurement objective function" and a "regularization objective function" when run in "Pareto" mode. However the user can make this association him/herself according to his/her definition of observation groups used in the Pareto process.

See section 4.25 of this document for a description of what PEST does when running in "Pareto" mode, and of its screen and file outputs. At any stage of a PEST run, a picture of the Pareto front (as traversed so far) can be obtained by plotting data contained in file *case.pod*; parameters corresponding to any point along the Pareto front are available in file *case.ppd* as well as in file *case.par.N*, where *N* signifies the iteration number and *case* signifies the filename base of the PEST control file.

2.43.4 Restarting a Pareto Run.

Normal re-start functionality is available when PEST runs in "Pareto" mode.

There will be many occasions however when a PEST run is finished and a user, on inspection of the outcomes of this run, would like to traverse more of the Pareto front than was traversed in the just-completed run. This can be achieved as follows.

1. Using the PARREP utility, create a new PEST control file containing the last set of parameters that PEST calculated on its previous run. These will correspond to the end of the Pareto front segment as traversed by PEST on its previous run. (Be sure to give the new PEST control file a different name to that of the old PEST control file so that the *pod*, *ppd* and *par* files from the previous run are not overwritten during the new run.)
2. Set PARETO_WTFAC_START to the previous value of PARETO_WTFAC_FIN, and PARETO_WTFAC_FIN to a suitable value. NUM_ITER_START can normally be set to zero to avoid repetition of the last iteration of the previous Pareto process.
3. Run PEST. The information in the *pod*, *ppd* and *par* files produced on this new run then supplements information contained in files of similar name produced on the previous run.

2.43.5 Special Considerations for SVD-Assist

If you wish to use SVD-assist for exploration of the Pareto front, this is easily accomplished. As is normal protocol for SVD-assisted inversion, the following steps should be taken.

1. Set up a Pareto PEST run using normal model parameters (following guidance provided above in selection of control settings and weights, and in defining observation groups).
2. Set NOPTMAX to -1 or -2 in this file.
3. Run PEST to produce a JCO file.
4. Run SVDAPREP in order to produce a PEST control file for SVD-assisted Pareto front exploration.

When PEST is run in "Pareto" mode when undertaking SVD-assisted inversion, the SVDA_MULBPA variable is automatically set to 1. Thus multiple BPA files are saved – one at the end of each iteration. These are named *base.bpa.0*, *base.bpa.1* etc, (where *base* is the filename base of the base PEST control file) and contain base parameter values corresponding to various points along the Pareto front as defined in the *pod* file (whose filename base corresponds to that of the super parameter PEST control file in the usual fashion). Under these circumstances, PEST does not leave a succession of *par* files, as the super parameter values that would be recorded in these files are of no real use to the modeler.

When undertaken SVD-assisted traversal of the Pareto front, parameter values recorded in the *ppd* file are actually super parameter values. This makes it easy for the user to link points on the Pareto front (as defined by objective functions listed in the *pod* file) to parameter values as actually used by the model. The filename base of the *ppd* file is the same as that of the super parameter PEST control file.

An extra setting is available when implementing SVD-assisted exploration of the Pareto front. This is supplied through a control new variable named SVDA_PAR_EXCL. This variable is placed within the "svd assist" section of the PEST control file immediately following the SVDA_SUPDERCALC variable on the third data line of this section; thus it is the fifth item on this line. SVDA_PAR_EXCL must be set to 0, 1 or -1. If it is set to 1, super parameters are defined using only sensitivities of model outputs that lie within the observation group whose weight factor is adjusted during the Pareto front exploration

process; this is the observation group whose name is provided as the PARETO_OBSGROUP variable in the "Pareto" section of the PEST control file. In the regularization context, members of this observation group are the measurements that are fitted during the calibration process. Prior information equations and other regularization observations that may be present within the base PEST control file are thus ignored in defining super parameters. Limited experience to date suggests that this variable is best set to 1 when using the Pareto process to enforce regularization constraints as described herein. If it is set to 0, super parameter definition is based on all observations and prior information equations cited in the base PEST control file (using the weights provided in that file).

If SVDA_PAR_EXCL is set to -1, then super parameters are computed on the basis of all observation groups in the PEST control file other than that named as the Pareto-adjustable group in the super parameter PEST control file. This is useful if traversal of the Pareto front takes place in the opposite direction – that is, from a fit which is initially too good (based on parameters which are too heterogeneous) to a fit which is not as good but which is based on more acceptable parameters (see below).

When SVDAPREP is used in order to build a super parameter PEST control file, it checks the setting of the PESTMODE variable in the base parameter PEST control file. If this is set to "Pareto" SVDAPREP asks the user for an appropriate setting for SVDA_PAR_EXCL. Its prompt (SVDAPREP's final prompt) is:

```
Provide setting for SVDA_PAR_EXCL [0, 1, or -1] (<Enter> if 0):
```

The default is 0 (for this is the best setting to use when using the Pareto concept to explore predictive uncertainty). Respond with "1" to set SVDA_PAR_EXCL to 1 in the super PEST control file written by SVDAPREP, "0" to set SVDA_PAR_EXCL to 0 in this file, or "-1" to set it to -1.

2.43.6 Going the Other Way

Suppose that you have just calibrated a model and that the calibrated parameter field contains too much variability. Suppose that you would like to remove some of this variability by slowly introducing stronger and stronger enforcement of regularization constraints, thereby maintaining as good a fit with measurements as possible while removing some of the more unsightly aspects of the calibrated parameter field. This is easily achieved by assigning all regularization prior information equations and regularization observations to a single observation group, and providing the name of that group as the PARETO_OBSGROUP variable in the "Pareto" section of the PEST control file. Initial parameter values should be those that provided "too good a fit" with the data. PEST can then reduce variability of this parameter field by moving towards the end-point of the Pareto front where regularization observations dominate measurement observations.

In doing this, it may be helpful to remember that if prior information is added to a PEST control file after a JCO file has been computed for that file, the JCO2JCO utility will automatically add the sensitivities for the additional prior information when building the new JCO file for the new PEST control file.

If using PEST's SVD-assist functionality to traverse the Pareto front in this direction, consider setting SVDA_PAR_EXCL to -1 in the super parameter PEST control file.

2.44 Parallelisation of First Model Run

Parallelisation of model runs is now more the norm than the exception in modern modelling

environments. When using PEST to calibrate a model, parallelisation of model runs is easily implemented using either Parallel PEST (which is described in this document and in the main PEST manual) or BEOPEST (which is described in a document of its own).

One problem that has, in the past, been faced by PEST users who wish to parallelise model runs is that PEST always undertakes one model run in order to obtain model outputs corresponding to initial parameter values before it undertakes its first suite of parallel runs for the purpose of filling the Jacobian matrix. While this initial model run is underway, slaves are standing idle. This, of course, wastes time. Where runs are being undertaken on a computing cloud and costs are being accrued by underutilized slaves, it also costs money.

This problem has now been overcome. If either of Parallel PEST or BEOPEST is run using the "/p1" command-line switch, the first model run is undertaken in parallel with model runs required for the filling of the Jacobian matrix. ("p1" stands for "parameter values #1" or "initial parameter values").

Suppose that Parallel PEST is estimating 100 parameters. If its execution is instigated with the "/p1" switch, Parallel PEST will immediately inform the user that it will execute 101 model runs (more than this if central or five-point-stencil derivatives are being used). It will then report to the screen (and to its run management file) the number of runs that are finished as it receives news of their completion from its slaves. When all 101 runs have been completed it writes to the screen the value of the initial objective function, together with other information that it would normally record once the initial run was completed. It also announces that the Jacobian matrix for iteration 1 has been calculated. Shortly thereafter, it commences computation of parameter upgrades, and instigates a sequence of model runs based on different Marquardt lambdas.

If model runs are interrupted during computation of this initial run package (or at any time during a PEST run for that matter), Parallel PEST can be restarted using the "/s" command-line switch. Parallel PEST will then re-commence execution at the same spot at which its previous execution was interrupted; it will read the outcomes of previous model runs from its restart file. Parallel PEST should not, however, be restarted with both the "/s" and "/p1" command-line switches together. Parallel PEST will figure out for itself the contents of a previously interrupted run package, so the "/p1" switch is not required. Furthermore as the "/p1" switch is only pertinent to the first model run of the first iteration, it has no bearing on the behaviour of a restarted Parallel PEST run, other than on the nature of what it retrieves from the restart file that was recorded during its previously interrupted run; this information is provided in the restart file itself.

2.45 Option not to save the Jacobian

When using PEST in inversion settings involving thousands, or even tens of thousands, of parameters, accompanied by Tikhonov regularization on those parameters, the use of compressed Jacobian storage becomes essential. When working in contexts such as this, the user may wish to prevent regular storage of the Jacobian matrix in a series of over-written JCO files as:

- it can be time-consuming at it requires multiple accesses and decompression of the compressed Jacobian storage array;
- the JCO file may be very large.

Storage of a Jacobian matrix JCO file can be prevented using the JCOSAVE variable. This is recorded on the 10th line of the PEST control file following the ICOV, ICOR, IEIG and

optional IRES variables. Its position following these variables can be intermixed with other optional variables on this line, namely JCOSAVEITN, REISAVEITN and VERBOSEREC. Its value must be supplied as either "jcosave" or "nojcosave". The default setting (which prevails if it is not supplied) is "jcosave". Alternatively if a value of "nojcosave" is supplied, then PEST will not write a JCO file in the course of its execution.

2.46 Smaller Run Record File

Where parameter, observation and prior information numbers are high, the PEST run record file becomes voluminous - so voluminous that it is almost impossible for a user to obtain information from it.

PEST will write a much shorter run record file if the VERBOSEREC variable on the 10th line of the PEST control file is set to "noverboserec". The default value for this variable (i.e. the value that will prevail if it is not supplied at all) is "verboserec".

VERBOSEREC follows ICOV, ICOR, IEIG and the optional IRES variable on the 10th line of the PEST control file. It can be supplied together with other optional text variables that may follow the above integer variables, these being JCOSAVE, JCOSAVEITN and REISAVEITN. Its ordering with respect to these latter variables is arbitrary.

2.47 Indexed Prior Information

When estimating values for a large number of parameters (for example ten thousand or more parameters), the handling of prior information that provides Tikhonov constraints for these parameters may become problematical if compressed Jacobian storage is employed (i.e. if the MAXCOMPDIM variable in the "control data" section of the PEST control file is set to greater than 1). By its very nature, the Jacobian submatrix which holds this prior information will be sparse; hence considerable benefits are to be gained from storing it in compressed form. However there is a price to be paid for compressed storage; there may be considerable computational overhead required for storing and accessing elements of the Jacobian matrix that are stored in this way.

Considerable computational advantage can be gained if prior information is provided to PEST in the same order as that in which it is stored internally in the compressed Jacobian array. Elements of the compressed Jacobian matrix are stored in column order, with zero-value elements excluded. That is, elements are stored in the order $\mathbf{X}(i,j)$, $\mathbf{X}(i+1,j)$, $\mathbf{X}(i+2,j)$ etc, with the jump made to column $j+1$ when all values in column j have been stored. The row number is the observation (including prior information) number (i in this example), while the parameter number is the column number (j in this example).

In the "prior information" section of the PEST control file, the user supplies prior information in the form of equations. If more than one parameter is cited in any of these equations, the above ordering is not respected. Where many prior information equations are supplied in order to provide regularization for many estimated parameters, PEST may take a considerable amount of time to read and store this prior information when compressed Jacobian matrix storage is employed.

In order to overcome this problem, PEST now provides an alternative means through which the user can supply prior information. This is referred to as "indexed prior information" herein.

PEST is informed that prior information will be supplied in indexed form if the NPRIOR

variable in the "control data" section of the PEST control file is supplied as negative. The absolute value of NPRIOR must indicate the number of prior information equations that are featured in the current calibration problem (as usual). However the fact that this value is negative signals to PEST that prior information is supplied in indexed form in the "prior information" section of the PEST control file.

When the indexed prior information protocol is adopted, the "prior information" section of the PEST control file is subdivided into two sections. There are no headers between these subsections, so one just runs into the other. The first subsection should contain NPRIOR lines of data. Each line of this subsection must contain four entries. These are as follows.

- The name of the prior information equation (12 characters or less without quotes or blanks); this is the PILBL variable.
- The "observed value" of the prior information equation (a real number); this is the PIVAL variable.
- The weight associated with the prior information equation (a real number); this is the WEIGHT variable.
- The observation group to which the prior information equation belongs (12 characters or less citing a group that has already been named in the "observation groups" section of the PEST control file); this is the OBGNME variable.

Part of this subsection is exemplified below.

```
* prior information
pr_r1  0.0  1.0 regul_row
pr_r2  0.0  1.0 regul_row
pr_r3  0.0  1.0 regul_row
pr_r4  0.0  1.0 regul_row
pr_r5  0.0  1.0 regul_row
pr_r6  0.0  1.0 regul_row
pr_r7  0.0  1.0 regul_row
pr_r8  0.0  1.0 regul_row
pr_r9  0.0  1.0 regul_row
pr_r10 0.0  1.0 regul_row
pr_r11 0.0  1.0 regul_row
pr_r12 0.0  1.0 regul_row
pr_r13 0.0  1.0 regul_row
pr_r14 0.0  1.0 regul_row
pr_r15 0.0  1.0 regul_row
etc
```

Part of the first subsection of the "prior information" section of a PEST control file in which prior information is supplied in indexed format.

Following NPRIOR items supplied as above is the second subsection of the "prior information" section of the PEST control file. This subsection begins with a line containing a single integer, this specifying the number of lines to follow. This integer must equal or exceed NPRIOR; it will be referred to as NUMINDEX herein.

Each of the following NUMINDEX lines must contain three entries. These are, in order:

- a column number of the Jacobian matrix (an integer);
- a row number of the Jacobian matrix (an integer);
- the value of the element of the Jacobian matrix corresponding to the nominated row and column numbers.

A Jacobian matrix column number is obtained by counting parameters in order of their

appearance in the "parameter data" section of the PEST control file; fixed and tied parameters are omitted from this count. A row number is obtained by counting first observations, and then prior information equations, in order of their appearance in the PEST control file. For prior information equations the row number must exceed NOBS (where NOBS is the number of observations featured in the PEST control file), and must be less than or equal to NOBS+NPRIOR. It is the user's task to calculate these indices him/herself; normally a PEST control file which features indexed prior information will be written by utility software which undertakes these calculations as part of its processing services.

The value of the Jacobian element is the factor by which the parameter (or log of the parameter if it is log-transformed) is multiplied in the respective prior information equation. This is equivalent to the sensitivity of the outcome of the equation to the parameter (or log of the parameter).

An example of the start of the second subsection of an indexed "prior information" section of a PEST control file follows.

```
    89544
        1      1090    1.000000
        1     16040    1.000000
        1     30990    1.000000
        2      1090   -1.000000
        2      1091    1.000000
        2     16065    1.000000
        2     31012    1.000000
        3      1091   -1.000000
        3      1092    1.000000
        3     16090    1.000000
        3     31034    1.000000
        4      1092   -1.000000
        4      1093    1.000000
        4     16115    1.000000
        4     31056    1.000000
        5      1093   -1.000000
        5      1094    1.000000
        5     16140    1.000000
```

Start of the second subsection of the "prior information" section of a PEST control file in which prior information is supplied in indexed format.

The fact that prior information is supplied in indexed format does not, of itself, guarantee fast compressed storage of this information, though it does help a little. What makes all the difference is the order in which the user supplies items comprising the second subsection of the indexed "prior information" section of the PEST control file. These should be supplied in the same order as that in which they will be stored internally by PEST. Thus the observation number must vary most quickly (second column of the above figure) while the parameter number (first column of the above figure) must vary most slowly; numbers in the first column should never decrease.

2.48 Binary External Derivatives File

The JACFILE variable in the "control data" section of the PEST control file must be set to a non-zero value if a model supplies PEST with derivatives which it calculates itself. These derivatives are supplied through an "external derivatives file", the name of which is recorded in the "derivatives command line" section of the PEST control file, along with the command

that PEST must use to run the model when used to calculate derivatives. If JACFILE is set to 2, the external derivatives file must observe JUPITER formatting protocol. If it is set to 1, then either of two PEST-specific formats can be adopted for conveyance of derivatives. These are described in the main PEST manual. In one of these formats an entire matrix is supplied. Alternatively, this matrix can be supplied in compressed form in which Jacobian elements are indexed and zero-valued elements are omitted. PEST detects whether the entire matrix is supplied, or whether an indexed, compressed matrix is supplied, through the entries provided on the first line of this file.

With recent changes to PEST this file can also be provided in binary format. Both full and compressed matrix storage options are supported.

As explained in the main PEST manual, if the derivatives matrix is supplied in ASCII format, the first line of the external derivatives file must contain either two or three variables. The first is NPAR, the number of parameters featured in the PEST control file, while the second is NOBS, the number of observation featured in the PEST control file. PEST ensures that these are in accordance with its expectations. Then follows an optional variable named NDIMCOMP. The presence of this variable signals to PEST that derivatives will be supplied in compressed format. Its value indicates the number of lines to follow. Each of these lines contains a parameter index, followed by an observation index followed by the sensitivity value associated with the Jacobian matrix element defined by these indices.

Optional variables are not possible in binary files. Hence, irrespective of whether a full or compressed matrix is provided, the NDIMCOMP variable must be supplied in the header to a binary external derivatives file. If NDIMCOMP is set to zero, PEST takes this as a signal that the full matrix follows. If it is set to a positive number, this indicates the number of ensuing data lines. If it is supplied as a negative number, PEST ceases execution with an error message.

A JACFILE value of -1 in the "control data" section of the PEST control file conveys to PEST that the model will provide derivatives in binary format. The protocol is the same as for ASCII format. The following of PEST's expectations should be noted:

- Integer values are expected as four byte numbers. However NCOMPDIM is expected to be of eight bytes length if an INTEL compiled version of PEST is employed.
- Derivative values are expected as double precision (eight bytes).

2.49 Working with Large Numbers of Parameters

A number of alterations have been made to PEST to expedite its performance in contexts where a very large number of parameters must be estimated. Some of these are documented in previous subsections. The purpose of the present subsection is to provide some recommendations for use of PEST in these contexts. These recommendations make use of features that have (at the time of writing) been recently added to PEST as documented in previous subsections, as well as features that have been available in PEST for a longer period of time.

2.49.1 Solution Methodology

Where parameters number more than a few thousand, LSQR becomes the only feasible method provided by PEST for solving the inverse problem. It provides numerical stability at the same time as it provides speed. Furthermore, its internal workings are such that it does not require access to all of the Jacobian matrix at the one time, for it requires access to only a row

or column of this matrix at any one moment of its processing sequence. This makes its use in conjunction with compressed internal storage of the Jacobian matrix relatively easy.

2.49.2 Derivatives

Where a large number of parameters is being estimated (in the tens of thousands) it becomes mandatory that derivatives be calculated by the model itself using methodologies which have been developed specifically for this purpose such as adjoint techniques. Use of the finite-difference methodology for calculation of derivatives becomes impossible under these circumstances for two reasons. These are:

- the large number of model runs that this would require, and
- the lack of numerical precision that accompanies finite-difference calculation of the sensitivity of each (relatively insensitive) individual parameter where dense parameterization schemes are employed within a model domain.

As discussed in the previous section, a model can supply self-calculated derivatives to PEST using an external derivatives files. Disk space and reading time can be saved if the following options are taken in supplying this file:

- use of compressed, indicial, formatting of sensitivities;
- binary storage.

If using the compressed file option, and if using compressed Jacobian matrix storage within PEST itself (see below), maximum efficiency is achieved if indexed Jacobian matrix elements listed in the external derivatives file are cited in the same order as that in which they are stored internally by PEST. Storage order within PEST is column by column; hence indexing progresses firstly by observation, and then by parameter.

A JACFILE value of -1 in the "control data" section of the PEST control file conveys to PEST that the model will provide derivatives in binary format. The protocol is the same as for ASCII format. The following of PEST's expectations should be noted:

- Integer values are expected as four byte numbers. However NCOMPDIM is expected to be of eight bytes length if an INTEL compiled version of PEST is employed.
- Derivative values are expected as double precision (eight bytes).

2.49.3 Compressed Internal Jacobian Matrix Storage

Compressed internal storage of the Jacobian matrix is instigated in PEST by supplying a positive value for the MAXCOMPDIM variable in the "control data" section of the PEST control file. Where many parameters are estimated, the size of the Jacobian matrix can become very large. Use of compressed storage may make the difference between this matrix fitting within a machine's memory or not. However as 64 bit systems become the norm, and as computer memory becomes larger, this is less of a problem than it used to be.

Nevertheless, in spite of the benefits offered by modern computing hardware, compressed Jacobian matrix storage may still be required where Tikhonov regularization is employed in the calibration process. The nature of regularization is such that at least one regularization constraint is normally required for each parameter. However in many instances more than one constraint may be employed per parameter - for example if smoothness constraints are applied in each of three directions by supplying prior information that specifies a preferred parameter value difference of zero between each parameter and its neighbour in each of these directions.

Where regularization is supplied through prior information equations, the Jacobian matrix is effectively divided into two submatrices. The first has dimensions of NOBS × NESPAR, where NOBS is the number of observations featured in the calibration dataset and NESPAR is the number of adjustable parameters. This submatrix contains the sensitivity of each model output used in the calibration process to each adjustable parameter. The second submatrix has dimensions of NPRIOR × NESPAR, where NPRIOR is the number of prior information equations that are featured in the current calibration exercise. This latter submatrix can be very large, and very sparse. In contrast, the first submatrix may not be as large, and will certainly not be as sparse.

To accommodate this common situation, PEST provides the ability to store the observation Jacobian submatrix (i.e. the first of the above submatrices) in a format that is effectively uncompressed, and the prior information Jacobian submatrix (i.e. the second of the above submatrices) in a format that is not only compressed, but that is internally indexed in a way that allows rapid access to neighbouring nonzero elements where nonzero elements are very sparse. This all takes place behind the scenes, transparently to the user, provided the value that he/she supplies for the MAXCOMPDIM variable is large enough.

It will be recalled that a value for MAXCOMPDIM which is greater than 1 signals to PEST that it must employ compressed Jacobian matrix storage. It also informs PEST of the size of the array that it must allocate to store the compressed Jacobian matrix; if the number of non-zero Jacobian elements exceeds this number, PEST will then cease execution with an appropriate error message. A user-supplied MAXCOMPDIM value is large enough to allow compressed storage of the type described above to take place (and to prevent Jacobian matrix storage overflow) if it exceeds a value of NOBS×NESPAR+(NPRIOR×NCITE)+10 where:

- NESPAR is the number of adjustable parameters;
- NOBS is the number of observations comprising the calibration dataset;
- NPRIOR is the number of prior information equations; and
- NCITE is the number of parameters cited in each prior information equation.

It is the user's responsibility to supply a value for MAXCOMPDIM which is equal to, or exceeds, this value if he/she would like differential submatrix storage of the type described above to be activated. If he/she does this, and if LSQR is employed as a solution device for the inverse problem of model calibration, this storage mechanism is automatically implemented. For the user's information, this fact is then recorded on the PEST run record file.

2.49.4 Other Labour-Saving Devices

Gains in speed in reading and storing the PEST input dataset can be had if prior information is supplied in indexed format, and in the correct order, as was explained in a previous subsection of this document.

Particular when using compressed internal Jacobian matrix storage, further gains in speed can be had if unnecessary access to the Jacobian matrix is kept to a minimum. Savings in this respect can be made if the RSTFLE variable in the "control data" section of the PEST control file is set to "norestart", and if the SAVEJCO variable is set to "nosavejco". In addition to this, the PEST run record file can be made considerably more readable if data which will never be read is not recorded on this file, this being ensured if the VERBOSEREC variable is set to "noverboserec".

Set the ICOV, ICOR, IEIG and IRES variables to 0 so that PEST makes no attempt to

compute the matrices invoked by these variables.

Experience suggests that using an initial Marquardt lambda value (RLAMBDA1) of 0.1 and a context-sensitive lambda multiplication factor (instigated using a negative RLAMFAC setting such as -3), can also increase the speed of the inversion process in very highly parameterized contexts.

2.49.5 Summary

The following is a summary of options which can improve PEST's performance in very highly parameterized contexts.

- Use the LSQR solution mechanism.
- If employing Tikhonov regularization, use compressed Jacobian matrix storage. Set MAXCOMPDIM to a value greater than NOBS×NESPAR+(NPRIOR*NCITE)+10.
- If using compressed Jacobian matrix storage then:
    - supply prior information in the PEST control file using indexed formatting in the correct order;
    - supply external derivatives in compressed (and preferably binary) form in correct order.
- Set RSTFLE to "norestart";
- Set JCOSAVE to "nojcosave";
- Set VERBOSEREC to "noverboserec";
- Set RLAMBDA1 to 0.1 and RLAMFAC to -3;
- Set ICOV, ICOR, IEIG and IRES to 0.

2.50 Iteration and Run Specific Parameter Values

The 10th line of the PEST control file (i.e. the 9th line of the "control data" section of the PEST control file) can now contain an optional variable named PARSAVEITN. If supplied, this variable must be recorded as either "parsaveitn" or "noparsaveitn". In the former case PEST will save a parameter value file at the end of every iteration of the parameter estimation process, this containing best parameter values achieved during that iteration, irrespective of whether they improved the overall objective function or not. This file is named *case*.*par*.*N* where *case* is the filename base of the PEST control file and *N* is the iteration number. A "parameter history" of the parameter estimation process is thus recorded. Alternatively, if PARSAVEITN is set to "noparsaveitn" or omitted altogether, an iteration-specific parameter value file is not recorded. Neither option interferes with PEST's normal behaviour of recording and refreshing a parameter value file (named *case*.*par*) containing best parameters achieved to date at the end of every optimisation iteration.

PARSAVEITN can be supplied at any position on the 10th line of the PEST control file following the ICOV, ICOR, IEIG and optional IRES variables. Its position with respect to other optional variables on this line (namely JCOSAVEITN, JCOSAVE, REISAVEITN and VERBOSEREC) is arbitrary.

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

If PARSAVERUN is set to "parsaverun" then PEST creates a series of parameter value files

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

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

2.51 Absolute Parameter Change Limits

As is documented elsewhere, parameters within a PEST control file can be declared as "factor limited" or "relative limited". In either case, a single variable governs their maximum allowed change on any one iteration of the parameter estimation process. The variable FACPARMAX designates the maximum factor change which any factor-limited parameter is allowed to undergo, while the variable RELPARMAX designates the maximum relative change which any relative-limited parameter is allowed to undergo. If the change which PEST calculates for any factor- or relative-limited parameter exceeds this value, PEST shortens the length of the overall parameter upgrade vector so that this does not occur.

Upgrade magnitudes for adjustable parameters may now be suppressed using absolute limits. As presently programmed, ten such limits can be provided. Any parameter can designated as being governed by one of these limits.

The following figure shows the "control data" section of a PEST control file. The seventh line of this file contains values for the RELPARMAX, FACPARMAX and FACORIG variables. Following that, the values for three absolute change limits are supplied (in bold).

```
pcf
* control data
restart estimation
19 19 2 0 2
2 3 single point
5 2 0.3 0.03 10
5 5 0.001 absparmax(3) = 1  absparmax(4)=0.5 absparmax (2)=1.0
0.5
50 0.01 3 3 0.01 3
```

The "control data" section of a PEST control file.

The following points define the protocol for provision of ABSPARMAX absolute change limits.

- The value of an ABSPARMAX change limit is supplied using the string "absparmax(*n*) = *r*" where *n* is an integer between 1 and 10 (as presently programmed) and *r* is a real number greater than 0.0. This string can be supplied in upper or lower case. Spaces can be inserted anywhere within the string.
- ABSPARMAX strings can be placed anywhere on the seventh line of the PEST control file. They are processed and removed before any other variables are read from this line. Other variables that can occupy this same line are RELPARMAX, FACPARMAX, FACORIG, UPVECBEND and IBOUNDSTICK. The protocol for reading all of these variables remains unchanged.
- Values can be supplied for any number of ABSPARMAX(*n*) variables, up to a maximum of 10 (as presently coded). However values for ABSPARMAX variables with the same *n* must not be repeated.

The following figure shows part of the "parameter data" section of a PEST control file.

```
* parameter data
ro1  none absolute(3) 1.000000 0.1 10000 ro 1 0
ro2  none absolute(2) 1.000000 0.1 10000 ro 1 0
ro3  none absolute(3) 1.000000 0.1 10000 ro 1 0
ro4  none absolute(3) 1.000000 0.1 10000 ro 1 0
ro5  none absolute(5) 1.000000 0.1 10000 ro 1 0
ro6  none absolute(5) 1.000000 0.1 10000 ro 1 0
```

Part of the "parameter data" section of a PEST control file.

In the above example the string "absolute(*n*)" replaces "relative" or "factor" as the value of the PARCHGLIM variable, this designating the pertinent parameter as being absolute-limited. The value of its absolute limit is that ascribed to ABSPARMAX(*n*) in the "control data" section of the PEST control file, where *n* is the index following "absolute" in the above designation. The following aspects of this protocol should be noted.

- If a parameter is tied or fixed, its absolute change limit is ignored.
- Parameters that are log-transformed cannot be assigned an absolute change limit; they can only be assigned a factor change limit.

If any parameter changes are absolute-limited, PEST prints out the maximum parameter change pertaining to any absolute change limit group to the screen and to the PEST control file at the end of every optimisation iteration.

2.52 User-Supplied Run Descriptor

At the start of any PEST run, the user can provide a short text string which is then recorded near the top of the run record file, and is written to the screen during each optimisation iteration. Where a sequence of PEST runs is being undertaken (for example as part of null space Monte Carlo based generation of calibration-constrained random parameter fields), this allows a user to know the position in the sequence of the PEST run that is being currently undertaken.

The user-supplied text string is provided through the command line following a "/t" command line switch. Suppose, for example, that the required text string is "My text". Then this can be provided to PEST by running it as follows.

```
pest case1 /t "my text"
```

(Note that when run using the above command, PEST reads the PEST control file *case1.pst*.)

The following aspects of the use of a text run descriptor should be noted.

- The descriptor text must be surrounded by double quotes.
- It must immediately follow the "/t" switch.
- If a "/t" switch is provided without an ensuing text run descriptor, an error message will be provided, and PEST will cease execution.
- Other switches can appear on the PEST command line together with the "/t" switch and run descriptor. These can either precede or follow the "/t" switch and run descriptor text. However the "/t" switch must not follow the "/h" switch and ancillary information required for the running of BEOPEST, as these must always appear last on the PEST command line.
- The run descriptor supplied for a restarted PEST run can be different from that supplied for the original PEST run. It is run-specific rather than case-specific.

- If the "/t" switch and text run descriptor are omitted from the command line, then no mention is made of the (blank) run descriptor, either on the screen or on the run record file.

The run descriptor is written to the top of the run record file. It is also recorded on the run record file immediately following a re-commencement of the recording of that file following a restart. It is written to the screen at the start of each optimisation iteration. See the following example.

```
 OPTIMISATION ITERATION NO.         : 7
    User run reference text         : "My text"
    Model calls so far              : 117
    Current regularisation weight factor                    :   1.8347
    Current value of measurement objective function         :   1.0144
    Current value of regularisation objective function      :   1.4688

    Starting phi for this iteration                         :   5.9583
    Contribution to phi from observation group "obsgp1"     :  0.53622
    Contribution to phi from observation group "obsgp2"     :  0.47816
    Contribution to phi from observation group "regul"      :   4.9440
```

Text written to the screen at the start of a new optimization iteration.

2.53 Multiple Model Commands and Parallelization

If a PEST input file employs multiple model command lines, BEOPEST (but not Parallel PEST) will respect this. When sending a parameter set to a slave it will also send the index of the model command that must be used for that particular model run. The slave will then use the appropriate command to undertake the model run. It is up to the user to ensure that all files necessary for undertaking all model runs are accessible from all slave working directories.

It will be recalled from PEST documentation that the NUMCOM control variable needs to be set to greater than 1 in the "control data" section of the PEST control file if PEST is to employ multiple model command lines. Command line indices specific to different parameters are provided through the DERCOM variable appearing in the "parameter data" section of the PEST control file. Recall that the command with index 1 is always employed for the initial model run and when testing parameter upgrades.

It should be noted that the use of multiple model commands is incompatible with use of SVDA as the linkage of commands to parameters embodied in the DERCOM array does not make sense if super parameters are used in place of base parameters, because the former are combinations of the latter.

2.54 Observation Re-referencing

2.54.1 Introduction

The use of so-called "observation re-referencing" adds considerable flexibility to the way that PEST can run a model. It allows PEST to run different variants of a model for different purposes. As such, it is an expansion of existing PEST functionality embodied in its ability to employ multiple command lines. However observation re-referencing goes well beyond existing PEST functionality, allowing at least the following strategies to be employed in model calibration and uncertainty analysis.

- Use of a simple model, or multiple simple models, in place of a complex model for the purpose of finite-difference derivatives calculation. Meanwhile parameter upgrades are tested and applied on a complex model.
- The ability to alter initial model conditions (for example initial heads in a steady state groundwater model) as parameters are upgraded. Where initial conditions can be made to approximate final conditions, the convergence time of iterative model solvers can be increased dramatically if these conditions are re-calculated by the model on the basis of most recently estimated parameters.

2.54.2 General Principles

PEST calculates derivatives of model outputs with respect to adjustable parameters using finite differences. In some cases it may be possible for a different model to be employed for the purpose of derivatives calculation from that which is used for the purpose of testing and upgrading parameters. This may be desirable where a simplified "surrogate model" is run for the purpose of derivatives calculation, and this model has a short run time compared with that of the model which is actually being calibrated. However if this strategy is attempted, certain problems may arise. These will now be discussed.

The difference equation through which the derivative of observation *j* with respect to parameter *i* is calculated can be written as follows:

$$\frac{\partial o_j}{\partial p_i} \approx \frac{o_j^{vi} - o_j^{r}}{p_i^{v} - p_i^{r}}$$

where p^r_i is the base value (i.e. the current reference value) of parameter p_i and o^r_j is the value of the model output corresponding to observation o_j calculated by the model using the base parameter set. Meanwhile p^v_i is the value of parameter *i* when incrementally varied for the purpose of derivatives calculation and o_j^{vi} is the model-calculated value of observation *j* when parameter *i* is thus varied.

Normally o^r_j is calculated by PEST during either the initial model run, or during the parameter upgrade process where different model runs are undertaken on the basis of different trial parameter sets calculated using different Marquardt lambdas. In the latter case the best upgrade is selected and new parameters are based on that; meanwhile model outputs calculated on the basis of the best parameter set become the base model outputs or "reference model outputs" used in the above equation for finite-difference derivatives calculation for the next optimisation iteration.

A problem arises if a different model (for example a simpler model) is to be employed for derivatives calculation from that which is used for testing parameter upgrades. In this case reference model outputs calculated using the complex model are not appropriate for use in the finite difference derivatives equation when applied to outputs of the simple model. The above equation is only valid when the same model is used to calculate both o^{vi}_j and o_j^r. Hence, for finite-difference-calculated derivatives to have integrity in this case, the simple model must be run especially to calculate o^r_j using the reference parameter set. This requires a special model run to be undertaken at the beginning of each iteration. This special model run is indeed carried out if "observation re-referencing" is activated.

In more complex cases where more than one version of the model is used for the purpose of derivatives calculation (for example if multiple simple models are employed, each used for the calculation of derivatives with respect to a different subset of parameters), then an

observation re-referencing model run must be undertaken for each version of the model that is so employed.

Using PEST's multiple command line functionality it has been possible for a long time to employ different model versions for calculation of derivatives with respect to different parameters. However in the past it has always been assumed that the different models employed for this purpose are all components of the one large model. Under these circumstances observation re-referencing is not required, for the same reference model output set applies to all of these. However with implementation of observation re-referencing it is no longer assumed that models used in finite-difference derivatives calculation produce the same outputs as the main model if supplied with the current reference parameter values, as has been required until now.

2.54.3 Observation Re-referencing - Mode 1

This mode of PEST's observation re-referencing functionality is automatically activated if only one command is used to run the model. Thus the NUMCOM variable must be set to 1 in the "control data" section of the PEST control file, or can be omitted altogether. At the same time, if the DERCOM variable appears in the "parameter data" section of the PEST control file, it must be assigned a value of 1 or 0 for each adjustable parameter.

If NUMCOM is set to 1 (or omitted) a single model command is provided in the "model command line" section of the PEST control file. As usual, this command may refer to an executable program, or to a batch file which cites many different executable programs which are run in succession. Where observation re-referencing is activated, the user must prepare not just the program or batch file cited in the "model command line" section of the PEST control file; he/she must supply two other programs or batch files as well. These are not mentioned by name in the PEST control file. However their names are assumed to be formed from the command cited in the PEST control file by addition of an "r_" prefix and a "d_" prefix to that name. Thus if the command cited in the "model command line" section of the PEST control file is *command.bat*, then PEST will, at various stages of its execution, issue three different commands to the operating system in order to run the model, these being *command.bat*, *r_command.bat* and *d_command.bat*.

These different model commands are used by PEST in the following manner.

- The command provided to PEST in the "model command line" section of the PEST control file (*command.bat* in the above example) is used by PEST to run the model for the first time, and to undertake attempted upgrades of parameters on the basis of different Marquardt lambdas. Outputs of this model are used to calculate the objective function that is written to the screen and recorded on PEST output files. These are also the model outputs that are recorded in the run record file and in the residuals file.
- The command with the "d_" prefix (*d_command.bat* in the above example) is used for the purposes of finite difference derivatives calculation. This command is issued at least once for every adjustable parameter during the Jacobian calculation phase of every iteration.
- The command with the "r_" prefix (*r_command.bat* in the above example) is used by PEST as it undertakes one additional run of the model during each iteration. This run is undertaken just prior to running the model at least once for every adjustable parameter for the purpose of filling the Jacobian matrix (i.e. for derivatives calculation). The parameters that are employed for this run are the current reference values of the parameters - these having just been upgraded during the previous

iteration. Model outputs calculated on the basis of this command are used as model output reference values for the purpose of derivatives calculation. Thus this command is used to compute o^r_j featured in the above equation for all *j*.

In many calibration circumstances the batch file which uses the "d_" prefix will have identical contents to that which uses the "r_" prefix. Thus reference model outputs are computed using the same version of the model as are outputs calculated using incrementally varied parameters for the purpose of derivatives calculation. On other occasions the re-referencing command may be more complex than this. For example if it is undertaken for the purpose of providing a revised set of initial heads as discussed above (and the model used for derivatives calculation is the same as that used to compute the objective function), the "r_" batch file may invoke commands which implement the following processing sequence.

- Computation of model outputs on the basis of current reference parameters values (which presumably have just been updated).
- Copying of domain-wide heads calculated on the basis of newly-upgraded reference parameters to an initial heads file.
- (If runs are parallelized), distribution of this initial heads file to all slave directories.
- Re-running the model once (still using newly-upgraded reference parameters) so that reference model outputs o^r_j are calculated under exactly the same numerical circumstances as those employed for calculation of model outputs on the basis of perturbed parameter values (this involving use of the new initial heads file).

2.54.4 Observation Re-referencing - Mode 2

This mode of observation re-referencing is automatically activated when PEST employs multiple model command lines. Thus NUMCOM is set to a number greater than 1 in the "control data" section of the PEST control file, and DERCOM variables associated with different parameters refer to different command lines. At the same time, more than one command is cited in the "model command line" section of the PEST control file.

In this case, as stated in existing PEST documentation, when PEST undertakes the initial model run, and when it undertakes parameter upgrades, it employs the first model command cited in the "model command line" section of the PEST control file, this being the command with index 1. When running the model for the purpose of derivatives calculation it can use any of the commands cited in the "model command line" section, this depending on the DERCOM value associated with each parameter.

When observation re-referencing is activated for a PEST case involving multiple command lines the above protocol is still followed. However PEST undertakes a number of additional model runs at the beginning of each iteration, just prior to undertaking model runs for the purpose of derivatives calculation. These are observation re-referencing runs.

Suppose that the commands listed in the "model command line" section of the PEST control file are *command1.bat*, *command2.bat*, *command3.bat*, etc. If observation re-referencing is activated, PEST will employ the commands *r_command2.bat*, *r_command3.bat* etc to undertake observation re-referencing runs just before undertaking finite difference Jacobian calculation using the commands *command1.bat*, *command2.bat*, *command3.bat* etc. It is the user's responsibility to ensure that the appropriate "r_" prefixed commands can be run; normally this will entail construction of appropriate batch files.

Note the following:

- An observation re-referencing model run is NOT undertaken for the first command employed in the "model command line" section of the PEST control file (i.e. *command1.bat* in the above example) because this same command was employed for the first model run and for the purpose of testing parameter upgrades. Reference model outputs calculated using this command are thus available, and are appropriate for finite-difference derivatives calculation for those parameters (if any) for which the first model command is run on the basis of incrementally varied parameters (i.e. for which the parameter-specific DERCOM setting is 1).
- Where a particular model command is employed to calculate finite difference derivatives for a particular parameter, the reference model outputs employed in the finite difference equation are those computed using the corresponding "r_" prefixed model command.
- If no parameters are assigned to a particular model command (i.e. if no parameter DERCOM value provides the index for that command), then that command is not employed in derivatives calculation. An observation re-referencing run is not then undertaken using the pertinent "r_" prefixed command.

2.54.5 Activating Observation Re-referencing

Observation re-referencing is activated by adding the string "obsreref" to the fifth line of the PEST control file. This is the same line as that on which the NTPLFLE, NINSFLE, PRECIS, DPOINT variables, and the optional NUMCOM, JACFILE and MESSFILE variables appear. The "obsreref" string can be placed anywhere on the line. Observation re-referencing can be de-activated (the default condition) by omitting this string, or by citing the string "noobsreref" on this line.

2.54.6 SVDA and Observation Re-referencing

SVDAPREP has been modified to read PEST control files in which observation re-referencing is activated. However it can only accommodate mode 1 of observation re-referencing. (Implementation of mode 2 would require that different commands be used to run the model for different parameters when undertaking SVD-assisted parameter estimation. However this is not possible as super parameters employed by PEST when undertaking SVD-assisted parameter estimation are each composed of many different base parameters. The relationship between parameters and model runs that is defined in a base parameter PEST control file cannot apply to super parameters.)

When SVDAPREP is run, it checks for the presence of "r_" and "d_" prefixed batch files that match the batch file supplied in the "model command line" section of the PEST control file. Then, in addition to the *svdabatch.bat* batch or script file, SVDAPREP writes files *r_svdabatch.bat* and *d_svdabatch.bat* by appending the commands to run PARCALC and, if necessary PICALC, to the contents of existing "r_" and "d_" prefixed batch files prior to any other commands cited within these files. The new PEST control file which is written by SVDAPREP instructs PEST to activate observation re-referencing. This file is immediately ready for running by PEST.

2.54.7 Parallelization

BEOPEST can be employed to run PEST on the basis of a PEST control file that implements observation re-referencing of either mode (including an SVDA PEST control file wherein mode 1 of observation re-referencing is activated). However Parallel PEST cannot be used

under these circumstances.

2.54.8 Restarting

Normal re-starting functionality is operative when observation re-referencing is employed. Use of the /i and /p1 command-line switches is also permitted.

2.55 Forgiving Derivatives Run Failure

2.55.1 General

A new variable named DERFORGIVE has been introduced to the "control data" section of the PEST control file. This has a similar role to the LAMFORGIVE variable which also features in that section. However whereas LAMFORGIVE governs PEST behaviour following model run failure when model runs are being undertaken for the purpose of testing parameter upgrades computed on the basis of different Marquardt lambdas, DERFORGIVE governs PEST's behaviour when model runs are being undertaken for the purpose of finite-difference derivatives calculation.

2.55.2 The DERFORGIVE Variable

The DERFORGIVE variable must be placed on the sixth line of the PEST control file. If present, it must follow the NUMLAM variable on that line. Its ordering relationship with other variables which can follow NUMLAM on this line (namely LAMFORGIVE and JACUPDATE) is arbitrary. It must be supplied as either "derforgive" or "noderforgive". If it is omitted, a value of "noderforgive" is assumed. A value of "noderforgive" activates normal PEST behaviour following model run failure during derivatives calculation. Under these circumstances PEST makes one attempt to repeat the model run with identical parameter values. If that run fails, PEST ceases execution with an appropriate error message.

2.55.3 Some Specifics

In normal operation, before PEST undertakes a model run it deletes all model output files for which there are corresponding instruction files. If any of those files are not present at the end of the model run PEST detects this condition as it tries to read them, reports it to the screen and to the run record file, and then ceases execution (after making one or more attempts to run the model).

When DERFORGIVE is set to "derforgive" PEST implements a different strategy for handling the absence of one or more model output files. Under these circumstances it reads those files that are indeed present. It writes a short message to the screen reporting the absence of each missing file. However it does not cease execution. Instead it assigns a derivative of zero with respect to the currently varied parameter to observations corresponding to model outputs that cannot be read because of the missing model output files.

Activation of DERFORGIVE functionality can be useful where different commands are used to run the model for calculation of derivatives with respect to different parameters. As stated in the preceding section, this situation may (or may not) be accompanied by activation of PEST's observation re-referencing functionality. In this case, those model batch files which are used for calculation of derivatives with respect to some parameters do not need to run model components (or file-copying surrogates for these components) which generate model outputs which are not sensitive to the pertinent parameters.

The BEOPEST version of PEST will accomodate a DERFORGIVE value of "derforgive". The slave will report missing model output files to the screen, and read the others. However Parallel PEST will not implement DERFORGIVE functionality; if Parallel PEST is run with DERFORGIVE set to "derforgive" it will cease execution with an error message.

2.56 Automatic PARLAM Setting

If NUMLAM is set to a negative number in the PEST control file, then if PEST is run as Parallel PEST or BEOPEST, the PARLAM variable will automatically be set to -9999. The number of parallel model runs used in the Marquardt lambda testing procedure is then equal to the absolute value of NUMLAM. If BEOPEST is run, this removes the need for a run management file in the working directory, the only role of this file being to reset PARLAM from its non-default value of 1. If a run management file is present, a negative NUMLAM value overrides the PARLAM value setting provided in that file.

2.57 Scaling using Parameter Bounds

2.57.1 General

Ideally, where subspace methods are employed for estimation of parameters it is not the actual parameters that should be estimated, but the parameters scaled by their innate variability. Where parameters show prior correlation (for example in accordance with a variogram) this should also be taken into account. Hence, ideally, parameters should undergo Kahunen-Loeve transformation prior to being estimated. See documentation of SSPLUS and SUPOBSPAR1 for further details of this transformation. (Note that the use of Tikhonov regularisation which describes pre-calibration parameter correlation through use of a covariance matrix achieves a similar thing.)

If a user supplies bounds for parameters which are a reflection of their pre-calibration variability (as they often are), these can be used as a surrogate for the pre-calibration prior probability distribution in determining the transformation which parameters should undergo for the purposes of their estimation. This description of their prior probability does not include pre-calibration correlation, but it many cases it will suffice. If the BOUNDSCALE parameter is set appropriately in the "control data" section of the PEST control file, PEST will actually scale parameters by their bounds interval (with logarithmic transformation taken into account) before estimating them. This is done under the following circumstances:

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

This is all done behind the scenes, invisible to the user. Hence no alterations to normal PEST usage, or to the viewing of PEST results, is required other than the setting of the BOUNDSCALE control variable. Furthermore, parameter bounds scaling can be undertaken regardless of whether Tikhonov regularisation is being employed or not. PEST adjusts its application of regularisation to accord with the transformed status of parameters as necessary.

2.57.2 Implementation of Bounds Scaling

Bounds scaling is implemented by adding the string "boundscale" to the eighth line of the PEST control file. This is the line (within the "control data" section of the PEST control file) that is led by the PHIREDSWH variable. The "boundscale" string can be placed anywhere on

the line following the PHIREDSWH variable, in any position in relation to other variables which can appear on this line; however it is suggested that it be placed at the end of the line to avoid confusion (omitting optional variables as desired). Alternatively, the string "noboundscale" can be written instead of "boundscale". If neither string appears on the line, this has the same effect as if "noboundscale" was written. The following figure depicts a "control data" section of a PEST control file in which the BOUNDSCALE variable appears.

```
* control data
restart regularisation
19 19 2 10 3
2 3 single point
10 -3 0.3 0.03 10
10 10 0.001 0
0.1 boundscale
20 0.01 3 3 0.01 3
1 1 1 1
```

The "control data" section of a PEST control file in which the BOUNDSCALE variable is featured.

The following should be noted.

- Obviously, if parameter bounds scaling is activated, then a user should try to ensure that a parameter's bounds are a good reflection of the uncertainty of that parameter. (For example, think of a parameter's upper and lower bounds as defining its 95% or 99% confidence interval.)
- Scaling of super parameters cannot be implemented. Hence if bounds scaling is activated in a super-parameter PEST control file through which SVD-assisted parameter estimation is undertaken, PEST automatically de-activates it. However it applies bounds scaling to calculation of super parameters from base parameters. This has the same effect as setting the SVDA_SCALADJ control parameter to 4 in the super PEST control file.
- If, for the purposes of SVD-assisted parameter estimation, bounds scaling is activated in the base parameter PEST control file, this will also be construed by PEST as an instruction to undertake base parameter bounds scaling in computation of super parameters. This has the same effect as setting the SVDA_SCALADJ control parameter to 4 in the super PEST control file.

3. Alterations and Additions to PEST Utilities

3.1 JCO2JCO

JCO2JCO has been upgraded so that the SCALE variable for some parameters can be altered between the original PEST control file for which a Jacobian matrix file (JCO file) already exists, and that for which it must be calculated. This can be very useful prior to undertaking SVD-assisted PEST runs where it is desired to reduce the sensitivity of non-log-transformed parameters such as recharge by decreasing its SCALE (and increasing its initial value by the same ratio) so that all parameter types have approximately the same composite sensitivity before commencing the inversion process.

JCO2JCO will allow a parameter to have a different scale on the second PEST control file from that which is cited on the first PEST control file provided the following conditions are met.

1. The parameter is not log-transformed in either file.
2. The parameter is not a tied parameter in either file.
3. The parameter has no parameters tied to it in either file.

JCO2JCO will not issue a warning message if a parameter's initial value is different between these two PEST control files as long as the product of the parameter's initial value and its SCALE is the same in both files.

The above restrictions may be such as to disallow certain complex alterations to parameter status between PEST control files. For example, a parameter cannot be given a different SCALE, and be assigned a different transformation or tied status between PEST control files, or JCO2JCO will object. If more complex changes in parameter SCALE and transformation status than those allowed on a single JCO2JCO run are required, this is not a problem, for JCO2CO can simply be run twice (or more) on the basis of more incremental changes between successively-altered PEST control files. Thus alteration of a parameter's SCALE and tied/fixed/transformation status becomes a two-step process rather than a single-step process.

3.2 PCLC2MAT

PCLC2MAT was written to help interpret data forthcoming from an SVD-assisted parameter estimation process.

Recall from the PEST manual that when PEST undertakes SVD-assisted parameter estimation, it estimates the values for a set of "super parameters", normally named *par1* to *parn*. However the model still "sees" a set of "base parameters"; these are the parameters whose values are actually written to model input files. However they are not written by PEST to these files; they are actually written by a utility program named PARCALC that calculates base parameter values from super parameter values as used by PEST.

Before each model run, PEST writes a PARCALC input file named *parcalc.in*. This file contains current values for super parameters, as well as the current "definition" of super parameters in terms of base parameters. This "definition" consists of the first NSUP

normalised eigenvectors of ($\mathbf{X}^t\mathbf{QX}$) or a similar matrix, where **X** contains sensitivities with respect to base parameters. Each component of a particular eigenvector is actually the contribution that the respective base parameter makes to the total super parameter. The ordering of parameters in an eigenvector is the same as the ordering of parameters provided to the base PEST control file upon which the SVD-assisted parameter estimation process was based. These base parameter names are also listed in *parcalc.in*.

PCLC2MAT is run using the command:-

```
pclc2mat parcalcfile ipar matoutfile
```

where

parcalcfile is a PARCALC input file (normally *parcalc.in*),

ipar is a super parameter number, and

matoutfile will contain the components of the *ipar*'th super parameter recorded as a single column matrix.

As is apparent from the above command-line syntax, the user-nominated column of the eigenvector matrix is written to a "matrix file"; the format of this file is described in a later section of this document. In the present case the second part of this file will list the names of base parameters (as matrix row names). The first section will provide the contribution made by each of these base parameters to the nominated super parameter; the squares of these contributions should sum to 1.

Alternatively, if *ipar* is supplied as a negative number in the PCLC2MAT command line, all of the first *ipar* super parameters are recorded in the matrix file written by PCLC2MAT.

As pointed out in the PEST manual, the base parameter composition of super parameters may alter during the SVD-assisted parameter estimation process. Such alterations occur if one or more base parameters hit their bounds. Such parameters are "frozen" at their bounds for the remainder of the parameter estimation process and are therefore not included in the definition of any super parameters. Singular value decomposition of $\mathbf{X}^t\mathbf{QX}$ is then repeated on the basis of the reduced number of adjustable base parameters, and super parameters redefined accordingly. Hence the super parameter definition recorded by PCLC2MAT will be pertinent only to that stage of the SVD-assisted parameter estimation process at which the identified *parcalc.in* was recorded. If it is at the end of the parameter estimation process, then it will correspond to the final definition of super parameters employed by PEST. However if no base parameters have hit their bounds during this process, then the super parameter definition contained in *parcalc.in* will pertain to all base parameters.

3.3 JCOPCAT

JCOPCAT concatenates two Jacobian matrices contained in two different unformatted "JCO" files written by PEST. Concatenation is carried out with respect to parameter values rather than observations; that is the matrices are concatenated "sideways" (assuming that each column pertain to a separate parameter and each row pertains to a separate observation) so that extra parameter columns are added to the file.

JCOPCAT can be very useful where it is desired that a single JCO file be built from the outcomes of two subsequent PEST runs based on different adjustable parameters but

featuring the same observations, for example where extra base parameters are being brought into play prior to an SVD-assisted PEST run. Use of JCOPCAT thus removes the need for re-calculation of sensitivities with respect to old parameters where new parameters are added to a base PEST control file. A new JCO file for the latter case can be built from that pertaining to pre-existing parameters and that pertaining to new parameters by running PEST (with NOPTMAX set to -1) based on only the new parameters and concatenating the old and new JCO files using JCOPCAT to form a complete JCO file based on all base parameters.

Use of JCOPCAT is predicated on the following assumptions regarding the two existing JCO files that are to be concatenated.

1. Both Jacobian matrices must contain the same number of rows.
2. Each row in each respective matrix must pertain to the same observation.
3. The same parameter cannot be featured in both JCO matrices.

Where these conditions are violated, JCO files can be prepared for concatenation using the JCOORDER utility. This may occur, for example, if prior information is featured in one JCO file but not in another. (JCO2JCO ignores such prior information when adapting the JCOPCAT-produced JCO file to an existing base PEST control file; thus it can be removed from one or both of the JCOPCAT input files before concatenation without any loss of information.)

JCOPCAT is run using the command:-

```
jcopcat jcofile1 jcofile2 jcofile3
```

where

jcofile1 is an existing Jacobian matrix file,

jcofile2 is another existing Jacobian matrix file, and

jcofile3 is a new concatenated Jacobian matrix file.

JCOPCAT reads both the *jcofile1* and *jcofile2* JCO files, reporting any errors or inconsistencies between the two to the screen. It writes the new, concatenated JCO file in unformatted form, ready for use by programs such as JCO2JCO. Note that the contents of a JCO file can be inspected in ASCII format using either of the JACWRIT or JCO2MAT utilities.

3.4 JCOORDER

JCOORDER reads an unformatted PEST-written "Jacobian matrix file" (i.e. "JCO file"). It then writes another JCO file after performing one or more of the following tasks on the PEST-generated Jacobian matrix contained in the first JCO file:-

1. removal of one or more rows of the Jacobian matrix (a row pertains to an observation);
2. removal of one or more columns of the Jacobian matrix (a column pertains to a parameter);
3. re-ordering of rows of the Jacobian matrix;
4. re-ordering of columns of the Jacobian matrix.

JCOORDER is run using the command:-

```
jcoorder jcofile1 orderfile jcofile2
```

where

jcofile1 is an existing Jacobian matrix file,

orderfile is a parameter/observation ordering file or PEST control file, and

jcofile2 is a new Jacobian matrix file written by JCOORDER.

An example of a "parameter/observation ordering file" is shown below:-

```
* parameters
ro2
ro3
h1
h2
* observations
ar3
ar4
ar5
ar6
ar7
ar8
ar9
ar11
ar12
ar13
```

Example of a "parameter/observation ordering file".

A parameter/observation ordering file must begin with the header line "***** parameters". Following that must be the names of one or more parameters, all of which must be cited in the first JCO file. However these parameters can be cited in any order; the order in which they are cited will be the order in which they will be represented in the final JCO file written by JCOORDER. Following parameter names, observation names must be presented in similar fashion.

The following should be noted:-

1. A blank line can be inserted at any location within a parameter/observation ordering file.
2. Any line beginning with the "#" character is ignored. Thus a comment can follow such a character.
3. Parameters and/or observations cited in the first JCO file can be omitted from the parameter/observation ordering file. These parameters/observations will then be omitted from the JCO file written by JCOORDER.

As an alternative to reading a parameter/observation ordering file, JCOORDER can read a PEST control file. It will recognise the fact that a PEST control file is supplied through the extension "*.pst*" supplied with this file on the JCOORDER command line. Ordering of parameters and observations will then be the same as that in the nominated PEST control file. The following should be noted:-

1. Tied and fixed parameters in the PEST control file are ignored.
2. Prior information in the PEST control file is ignored (however if prior information is

detected in the PEST control file, JCOORDER asks the reader to confirm that it is alright to ignore it).

3. Parameters/observations occurring in the JCO file can be absent from the PEST control file. However extra parameters and/or observations must not be present in the PEST control file.

Use of JCOORDER can be a necessary adjunct to the use of JCOPCAT in preparation for SVD-assisted parameter estimation where PEST runs are undertaken for the purpose of obtaining sensitivities for subsets of base parameters. The Jacobian matrices produced through this process will need to be concatenated into one big base Jacobian matrix. However it is quite possible that they will include different items of prior information; prior information is ignored by JCO2JCO and SVDAPREP when preparing for an SVD-assisted PEST run. Hence before parameter-concatenation of Jacobian sub-matrix JCO files using JCOPCAT, it will be necessary to remove the rows of these files which pertain to prior information, so that these matrices are compatible. Column re-ordering may also be necessary.

The easiest way to build a parameter/observation ordering file is to first run JCO2MAT on the Jacobian matrix requiring row/column removal and/or row/column re-ordering. Parameter and observation names are listed as column and row names respectively in this file. A little cutting and pasting with a text editor will soon result in a parameter/observation ordering file. Alternatively, a parameter/observation ordering file can be easily built from a PEST control file – perhaps the base PEST control file which the concatenated JCO file is being built to complement prior to running SVDAPREP.

3.5 JCOADDZ

JCOADDZ (the "Z" stands for "zero") is in some respects similar to JCOORDER. It reads a Jacobian matrix file and a file whose format is very similar to that of a parameter/observation ordering file. It also writes a new Jacobian matrix file. However instead of a parameter/observation ordering file, the file read by JCOADDZ is referred to as a "parameter/observation addition file"; furthermore the option of reading a PEST control file in place of this file is not available for JCOADDZ as it is for JCOORDER.

JCOADDZ is used for the purpose of adding additional rows (observations) and/or columns (parameters) to an existing Jacobian matrix file. All additional rows and columns are zero valued. The names of parameters pertaining to extra columns are read from the parameter/observation addition file, as are the names of observations pertaining to added rows.

JCOADDZ is run using the command:-

```
jcoaddz jcofile1 addfile jcofile2
```

where

jcofile1 is an existing Jacobian matrix file,

addfile is a parameter/observation addition file, and

jcofile2 is a new Jacobian matrix file written by JCOADDZ.

Note the following aspects of JCOADDZ's operations.

1. It is not essential that both parameters and observations be added to an existing

Jacobian matrix file. Thus the parameter/observation addition file may cite no observations or no parameters. However, irrespective of whether or not any parameters or observations are cited in each section of this file, the "* parameters" and "* observations" headers must still be present.

2. Blank and comment lines are allowed in parameter/observation addition file.
3. Extra parameters and observations are appended to the last column and row respectively of the existing Jacobian matrix file. If it is desired that they occupy different rows and/or columns, this can be accomplished through use of the JCOORDER utility.

3.6 GENLIN

3.6.1 General

"GENLIN" stands for "GENeral LINear model". The main purpose of GENLIN is to complement PESTLIN functionality through which a nonlinear model can be replaced by its linearised equivalent for purposes such as predictive uncertainty analysis. However it can also be employed on a stand-alone basis without the use of PEST at all.

GENLIN calculates its outputs according to the equation:-

$$\mathbf{y} = \mathbf{y}_0 + \mathbf{M}(\mathbf{p} - \mathbf{p}_0)$$

where **M** is a matrix, and **y**, $\mathbf{y}_0$, **p** and $\mathbf{p}_0$ are vectors. For the sake of conformity with PEST terminology (and in conformity with use of GENLIN as an adjunct to PESTLIN), the **p** vector will normally contain parameter values while the **y** vector will contain the model-generated equivalents to field measurements, or simply "observations" in PEST parlance. $\mathbf{p}_0$ and $\mathbf{y}_0$ are parameter and observation offsets.

The matrix **M** is a "sensitivity matrix", or Jacobian matrix of a linearised model. It must have as many rows as there are elements of **p** and as many columns as there are elements of **y**. GENLIN allows parameters to be log-transformed. In that case rows of the **M** matrix corresponding to log-transformed parameters must contain sensitivities with respect to the logs of respective parameters. Under these circumstances, GENLIN outputs are calculated as:

$$\mathbf{y} = \mathbf{y}_0 + \mathbf{M}[\log(\mathbf{p}) - \log(\mathbf{p}_0)]$$

where the log is taken to base 10.

3.6.2 GENLIN Input File

GENLIN requires an input file in which specifications for the linear model are supplied.. An example of a GENLIN input file is shown below. Such a file is easily prepared with a text editor.

```
* dimensions
5  19
* parameters
ro1  log     4.0   2.0
ro2  log     4.0   2.0
ro3  log     4.0   2.0
h1   none    2.0   0.0
h2   none    2.0   0.0
* observations
ar1    2.0
ar2    2.0
ar3    2.0
ar4    2.0
ar5    2.0
ar6    2.0
ar7    2.0
ar8    2.0
ar9    2.0
ar10   2.0
ar11   2.0
ar12   2.0
ar13   7.0
ar14   7.0
ar15   7.0
ar16   7.0
ar17   7.0
ar18   7.0
ar19   7.0
* sensitivities
ves1.jco
```

Example of a GENLIN input file.

A GENLIN input file must contain four sections in the order shown above. Each section must be named as shown, with the name preceded by the "*" character and a space. The contents of each of these sections are now described.

Dimensions

The line following the "dimensions" section header must contain two integers, these being the number of "parameters" and number of "observations" comprising the linear model. Once GENLIN has read these dimensions it can allocate array storage for the data to follow.

Parameters

There must be as many lines in this section of the GENLIN input file as the number of parameters specified in the "dimensions" section of the file. Each such line must contain 4 entries. The first is a parameter name (twelve characters or less in length), the second is its transformation state ("log" or "none"), the third is its current value (**p** in the above equations), while the fourth is its "offset value" ($\mathbf{p}_0$ in the above equations).

The following should be noted.

- In contrast to a PEST control file, parameters cannot be tied or fixed in a GENLIN input file; they can only be log-transformed or untransformed.
- Actual parameter and offset values, and not their log-transformed counterparts, must be supplied irrespective of the transformation status of a parameter Where parameters

are designated as log-transformed, GENLIN takes care of logarithmic transformation itself.

Observations

This section must contain as many lines as the number of observations specified in the "dimensions" section of the GENLIN input file. Each such line must contain two entries. The first is the observation name (twenty characters of less in length), while the second is the observation offset value ($\mathbf{y}_0$ in the above equations).

Sensitivities

The "sensitivities" section of the GENLIN input file must contain just one line, this being the name of a file containing the sensitivity of each linear model output with respect to each parameter (i.e. the elements of the matrix **M** in the above equations). If the filename extension is "*.jco*", GENLIN assumes that this matrix is stored in a PEST binary Jacobian matrix file (i.e. "JCO file"). If not, ASCII storage in matrix file format is assumed; the format of this kind of file is discussed in following sections of this addendum. Note that the JCO2MAT utility can be used to convert a JCO file to a matrix file.

It is important that the sensitivity file match the specifications for the linear problem as set out in the GENLIN input file. That is, the number of rows in the sensitivity matrix must be the same as the number of observations in the GENLIN input file. Similarly, the number of columns in the sensitivity matrix must be the same as the number of parameters in the GENLIN input file. Ideally, the names should match as well, though this is not essential – see below. Whether or not this is the case, the ordering of parameters and observations must be the same in both of these files.

It is also important to note that, as discussed above, if a parameter is cited as log transformed in a GENLIN input file, the sensitivity of that parameter to all observations provided in the sensitivity file must pertain to the log of that parameter. Fortunately this is easily accomplished if PEST is employed to calculate parameter sensitivities.

3.6.3 Running GENLIN

GENLIN is run using the command:-

```
genlin infile outfile [derivfile] [/c]
```

where:-

infile	is a GENLIN input file,
outfile	is a GENLIN output file,
derivfile	is an optional derivatives output file, and
"/c"	is an optional name-checking switch.

GENLIN writes its output file in matrix file format as a one-column matrix (i.e. as a vector). Observations retain the same names as those supplied in the GENLIN input file while the single column is provided with the arbitrary name *col1*.

Optionally, GENLIN can check that parameter names provided in the GENLIN input file match those provided in a JCO file or column names provided in an ASCII sensitivity matrix file. It can also check that observation names provided in the GENLIN input file match those

provided in a JCO file or row names provided in an ASCII sensitivity matrix file. It also checks that parameter and observation ordering is the same in both cases. This checking functionality is activated using the optional "/c" switch on the command line. GENLIN does not perform much checking in addition to this. In its normal role it will be run repeatedly by PEST as part of some kind of optimisation process. In this context, extensive, and possibly time-consuming, checks of all aspects of input data integrity on each occasion that GENLIN is run is unnecessary.

Another GENLIN option is the writing of a "derivatives file". This file is useful when GENLIN is employed in conjunction with PEST, providing it with "model-generated derivatives" as an alternative to finite-difference based derivatives, thereby saving PEST the need to carry out many GENLIN runs in the course of its execution. Note that, in accordance with PEST's requirements, derivatives written to this file pertain to native parameters rather than log-transformed parameters; hence where parameters are log-transformed the entries of this file will differ from those of the GENLIN input sensitivity file.

3.7 PESTLIN

3.7.1 General

Use of PESTLIN assumes that a complete parameter estimation (regularised or otherwise), or predictive uncertainty analysis, PEST input dataset exists. It also assumes that a Jacobian matrix exists for this dataset, together with a run record file in which parameter values pertinent to this Jacobian matrix are recorded. On the basis of this Jacobian matrix it writes a GENLIN input dataset, encapsulating a linearised form of the model. It also writes a new PEST control file so that (regularised) parameter estimation or predictive analysis can be undertaken on the basis of the linearised model. Because this linearised model will generally run much faster than the real model, concepts can be tested and/or approximate solutions to these types of problems can be obtained very quickly indeed.

In most cases the easiest way in which a linearised equivalent of a PEST input dataset can be built (including a linearised equivalent of the model) is as follows.

1. As stated above, build an entire PEST input dataset based on the real model. Ensure that this dataset is correct and consistent using the PESTCHEK utility.
2. Set NOPTMAX to -1 in the pertinent PEST control file.
3. Run PEST. PEST will undertake enough model runs to calculate the Jacobian matrix. Then it will cease execution, recording initial parameter values, together with uncertainty statistics (if these are calculable), on its run record file.

Alternatively, PESTLIN can be run at the end of a parameter estimation process. In this case the Jacobian matrix file will contain sensitivities with respect to optimised parameter values (or, if a minor improvement in parameter estimates occurred on the last iteration, with respect to near-optimised parameter values), while the run record file will record optimised parameter values, together with model-generated measurement equivalents calculated on the basis of these parameters (unless PEST is run using SVD-assist, in which case the PARREP utility should be employed to construct such a run record file by first constructing a new PEST control file on the basis of optimised parameters).

3.7.2 Using PESTLIN

PESTLIN is run using the command:-

```
pestlin pestinfile pestoutfile linbasename [/d]
```

where:-

pestinfile	is the filename base of a PEST input dataset,
pestoutfile	is the filename base of a PEST output dataset,
linbasename	is the filename base of a GENLIN linear model dataset, and
"*/d*"	(optionally) activates PEST external derivatives functionality.

PESTLIN undertakes the following tasks.

1. It reads the PEST control file (whose name is obtained by adding the extension ".pst" to *pestinfile* supplied on the PESTLIN command line) to obtain specifications for the current parameter estimation exercise.
2. It reads the corresponding Jacobian matrix file *pestinfile.jco* in order to obtain sensitivities of all adjustable parameters.
3. It reads the corresponding run record file *pestinfile.rec* in order to obtain the values of adjustable parameters on whose basis sensitivities were calculated (or, as described above, numbers which are close to these values), as well as corresponding model outputs. (These will become parameter and observation offsets in the GENLIN input dataset written by PESTLIN.)
4. It writes a GENLIN input file named *linbasename.in*, where *linbasename* is supplied on the PESTLIN command line as described above. Through this file GENLIN is instructed to write an output file named *linbasename.out*, and to read sensitivities of adjustable parameters from the file *pestinfile.jco*. If the /d switch is set on the PESTLIN command line, GENLIN is also instructed to write a PEST-compatible external derivatives file named *linbasename.drv*.
5. It writes an instruction file named *linbasenam.ins* through which GENLIN-generated observation values can be read.
6. It generates a template file named *linbasename.tpl* through which a GENLIN input file can be written by PEST based on current parameter values.
7. It writes a PEST control file named *pestoutfile.pst* which formulates the same parameter estimation problem as that formulated by the original PEST control file *pestinfile.pst*, but this time on the basis of a linearised model encapsulated in GENLIN.

PEST can then be run on the basis of *pestoutfile.pst*. If PESTLIN is run with the "/d" switch on its command line, PEST execution should be very fast. Even if it is not, and PEST then calculates linear model derivatives using finite differences, use of PEST on the linearised model should still be much faster than use of PEST in conjunction with the original model due to the fact that the former will probably have a vastly smaller run time than the latter.

Note that the GENLIN dataset constructed by PESTLIN does not make mention of fixed or tied parameters. Existence of the latter, however, is automatically taken into account through the fact that sensitivities of parent parameters are a function of the sensitivities of any parameters that are tied to them; their optimised values are thus also a function of these parameters. Note also that because tied and fixed parameters are missing from the PESTLIN-constructed PEST input dataset, optimised parameters recorded in the *pestoutfile.par*

parameter value file lack these parameters. Fortunately, a newly-modified PARREP can still be employed to insert parameter values found in ***pestoutfile**.par* into the original PEST control dataset encapsulated in ***pestinfile**.pst* to create a new PEST control file based on optimised parameters. In creating the new PEST control file, tied parameters are simply assumed to maintain the same ratio with their parent parameters as their initial values do. The fact that tied parameters are missing from the parameter value file thus does not affect their ability to be included in a new PEST control file based on the original (nonlinear) model, written by PARREP to reflect optimised parameter values.

3.8 PARREP

The role of PARREP is to replace initial parameters in a PEST control file with parameter values optimised on a previous PEST run. It reads a parameter value file produced during that run to obtain a set of new parameter values. These are then directly substituted for parameter values in the existing PEST control file in creating the new PEST control file.

Up until now, PARREP insisted that all parameters cited in the existing PEST control file be cited in the parameter value file. Now, however, it is more forgiving of omissions in the parameter value file. In particular, if a parameter is tied or fixed in the existing PEST control file, PARREP will not object if that parameter is omitted from the parameter value file. The value of a fixed parameter is simply transferred from the existing PEST control file to the new PEST control file. The value of a tied parameter is calculated from the new value assigned to its parent parameter on the assumption that the ratio between the two remains the same in new PEST control file as it was in the old PEST control file.

3.9 OBSREP

3.9.1 General

OBSREP does for observations what PARREP does for parameters. OBSREP reads optimised model outputs corresponding to observations and prior information from a "residuals file" produced as an outcome of a previous PEST runs. It then substitutes these values for "observed values" in the PEST control file. (It should be obvious from this that if both PARREP and OBSREP are run after completion of a PEST run, the objective function should be zero.)

3.9.2 Running OBSREP

OBSREP is run using the command:-

```
obsrep resfile pestfile1 pestfile2
```

where

resfile is the name of a "residuals file" written by PEST (extension "*.res*"),

pestfile1 is the name of an existing PEST control file, and

pestfile2 is the name of the new PEST control file to be written by OBSREP.

In most cases the residuals file *resfile* will have been produced by PEST on the basis of a run undertaken using *pestfile1* as the PEST control file. However this does not have to be the case. OBSREP will work correctly as long as every observation and prior information equation cited in the *pestfile1* PEST control file is also cited in the *resfile* residuals file. If *resfile* cites other observations and/or prior information items, and/or these are listed in a

different order in *resfile* from that prevailing in *pestfile1*, OBSREP will not object. However if any observations or prior information items cited in *pestfile1* are missing from *resfile*, OBSREP will cease execution with an appropriate error message.

3.10 PWTADJ1

3.10.1 General

One of the decisions that must be made prior to undertaking a parameter estimation run, is the weight to assign to different observations. In some cases weights should be assigned as the inverse of the uncertainty associated with each observation. However in other cases different weighting strategies should be employed. For example, if regularised inversion is being carried out, and if a particular observation, or group of observations, is more salient to a critical prediction required of a model than other observations, then there is justification for giving greater weight to the pertinent observation(s) in the parameter estimation process. See Moore and Doherty (2005) for a more complete discussion of this issue. In other cases it may be desirable that each type of observation be allowed to make a significant contribution to the estimation of model parameters, irrespective of how many actual observations comprise that data type, especially in instances (which can occur often in surface water model calibration) where a multi-component objective function is formed through processing raw observation data in different ways in order to extract from this data information pertinent to different parameters.

PEST allows observations (and prior information equations) to be divided into different groups. The contributions made to the total objective function by these different groups is written to both the screen and to the run record file at the beginning of the parameter estimation process, and at all stages of it. One often-used technique for determining appropriate relative weighting of different observation groups is to undertake the following procedure.

1. Build a PEST input dataset in which correct intra-group weighting has been set for all observation groups, but for which the weighting strategy between groups is yet to be determined.
2. Run PEST with the NOPTMAX control variable set to zero. PEST will then run the model just once and print to the screen, and to the run record file, the contribution made to the objective function by each observation group.
3. In the PEST control file, multiply all weights pertaining to each observation group by a group-specific factor such that the contribution made to the objective function by each observation group will be about the same as that made by every other group after these factors are applied.

If this procedure is followed then, at the beginning of the parameter estimation process at least, no observation group will dominate the objective function, nor will be dominated by other observation groups.

PWTADJ1 was written to automate this weights adjustment strategy.

3.10.2 Running PWTADJ1

PWTADJ1 is run using the command:-

```
PWTADJ1 casename pestoutfile contribution
```

where

casename is the name of an existing PEST case (that is, the filename base of an existing PEST control file);

pestoutfile is the name of a new PEST control file; and

contribution is a real, positive, number designating the desired contribution of each observation group to the overall objective function.

PWTADJ1 undertakes the following tasks:-

1. It reads the existing PEST control file, ascertaining the names of all observation groups cited in that file.
2. It reads the corresponding run record file, ascertaining the contribution made to the objective function by each observation group after the first model run (this being based on initial parameter values).
3. It writes a new PEST control file in which observation weights are adjusted such that, based on initial parameter values, the contribution made to the objective function by each observation group will be that desired by the user (i.e. the number supplied as *contribution* on the PTWTADJ1 command line).

Note the following aspects of PWTADJ1 operation.

1. *casename* supplied on the PWTADJ1 command line should not include the extension "*.pst*". However if it does include this extension, PWTADJ1 will remove it, assuming that *casename* is actually the filename base of this file. It will then read files *casename.pst* and *casename.rec*, on the assumption that the latter is the run record file corresponding to the former. If these two files are incompatible (PWTADJ1 checks only observation group names) PWTADJ1 will cease execution with an appropriate error message.
2. If the extension "*.pst*" is omitted from the name of the PEST control file to be written by PWTADJ1 (this being the second PWTADJ1 command line argument), PWTADJ1 will add this extension automatically.
3. It is the user's responsibility to run PEST on the basis of the existing PEST control file (preferably with NOPTMAX in that file set to zero) in order to generate a run record file in which initial objective function contributions are recorded.
4. If a covariance matrix is assigned to an observation group, PEST does not adjust weights for this group, as weights supplied in the PEST control file are not employed for this group; instead a weighting matrix is calculated on the basis of the supplied covariance matrix. In this case the user must multiply all elements of the supplied covariance matrix by a certain factor if objective function equalisation is to take place; this can be achieved using the MATSMUL utility documented below. PWTADJ1 writes the adjustment factor to the screen for each affected observation group.
5. PWTADJ1 adjusts weights for both observations and prior information.
6. If the PEST control file supplied to PWTADJ1 requests that PEST run in regularisation mode, PWTADJ1 will not adjust weights assigned to any regularisation group, that is any observation group whose name begins with "regul".

7. If the PEST control file supplied to PWTADJ1 requests that PEST run in predictive analysis mode, PWTADJ1 will not adjust the weight assigned to the single member of the observation group named "predict". (Be VERY careful when using PWTADJ1 on a predictive analysis PEST input dataset. You will certainly need to adjust the value of the PD0 predictive analysis control variable after using PWTADJ1. If PREDNOISE is set to 1, the weight assigned to the prediction may also require adjustment.)
8. The PEST input dataset should be checked using PESTCHEK before being supplied to PWTADJ1.
9. The PEST control file written by PWTADJ1 retains the same NOPTMAX setting as found in the original PEST control file. If NOPTMAX was set to zero in this file, it is important to remember to set it to a higher number before undertaking parameter estimation on the basis of the new PEST control file.

3.11 PWTADJ2

3.11.1 General

PWTADJ2 is similar in many respects to PWTADJ1. Its task is to adjust weights in a PEST control file. However the target objective function for which weights adjustment is sought is different from that sought by PWTADJ1. Like PWTADJ2, PWTADJ1 reads the current objective function, and the contribution made to that objective function by different observation groups, from a run record file associated with a nominated PEST control file; objective function details corresponding only to the initial model run are read from this file.

The aim of PWTADJ2 is to provide each observation with a weight that is the inverse of the standard deviation of noise associated with its measurement. On the assumption that the nominated PEST control file contains optimised parameter values (this can be ensured using the PARREP utility if desired), the expected value of the objective function corresponding to the first model run should then be equal to the number of non-zero observations contained within the PEST dataset. Similarly, the contribution to the objective function made by each observation group should then be equal to the number of non-zero weighted observations comprising the group. Similar considerations apply if a covariance matrix is supplied for an observation group instead of weights. In that case too, if the covariance matrix is correct, the contribution made to the objective function by the observation group should be equal to the number of observations in the group.

Actually, the situation is a little more complicated than this. If parameter values have in fact been estimated by PEST, then the expected value of the objective function is $n - p$, where n is the number of non-zero-weighted observations and p is the number of estimated parameters. Thus an observation weights adjustment process which multiplies all observation weights by a factor such that this occurs will ensure that these weights approximate the inverse of respective measurement error standard deviations. If any observation group employs a covariance matrix instead of weights, then all elements of the covariance matrix should be multiplied by the inverse square of the factor by which weights are multiplied in order to ensure that the total objective function achieves its desired value.

In achieving a target objective function of n or n-p, PWTADJ2 allows the user two options. All measurement weights can be multiplied by the same factor to achieve this total. Alternatively, a different multiplier can be employed for each observation group such that the

objective function contribution for each such group is equal to either n_g, the total number of non-zero-weighted members of the group, or $n_g(n-p)/n$; in the latter case the total objective function is *n-p*. This latter strategy may be employed in response to the fact that the parameter estimation process may encounter much more difficulty in fitting some observations types to field measurements over others. If this is taken as an indication of a greater-than-anticipated level of noise associated with the offending observation type, determination of a group-specific weight factor in this manner may allow calculated weights for that group to better approximate the inverse of noise standard deviation associated with measurements in the group. It should be noted however that while such a strategy may certainly prove useful, alteration of the relative weighting of different observation groups in this manner erodes the theoretical basis for selection of $n - p$ as the expected total objective function (though this matters little if p is small).

A further problem with the *n-p* concept arises where regularised inversion is undertaken and the number of estimated parameters is high (as it often is when the invere problem is solved through regularised inversion). In this case, even though values may be assigned to many parameters, the dimensionality of the solution space of the inverse problem may actually be quite small, so that the number of parameters that are *effectively* estimated may be little more than for a traditional over-determined parameter estimation problem. Where SVD is employed as the regularisation device, the dimensionality of the calibration solution space is equal to the number of pre-termination singular values. Where Tikhonov regularisation, or SVD-assist with a Tikhonov component, is employed, the dimensionality of the solution space can only be guessed.

Despite these limitations, PWTADJ2 can provide a very useful means of replacing weights in a PEST control file with those that are approximately inversely proportional to the standard deviation of measurement noise. This can be a useful exercise to perform prior to using one of the PREDVAR* utilities, or in building a PEST control file that is cited in an observation uncertainty file in construction of an overall observation covariance matrix.

3.11.2 Using PWTADJ2

PWTADJ2 is run using the command:-

```
PWTADJ2 pestinfile pestoutfile use_groups [parameter_correction]
```

where:-

pestinfile is the name of an existing PEST control file (omit the “pst” extension if desired);

pestoutfile is the name of the new PEST control file which PWTADJ2 will write;

use_groups must be supplied as “g” or “ng” signifying “groups” or “no-groups” (this will be further discussed below); and

parameter_correction is the p of the $n - p$ term; if this item is omitted from the command line, p is assumed to be zero.

Upon commencement of execution, PWTADJ2 reads the nominated PEST control file. Then it reads the associated run record file. Thus it is assumed that PEST has been run on the basis of nominated PEST control file. However PWTADJ2 only reads objective function information pertaining to the first model run from the run record file. Hence in most cases that PWTADJ2 is used, the NOPTMAX control variable in the PEST control file will have been set to 0 so that only one model run is carried out by PEST. Furthermore, as suggested

above, on many occasions the PARREP utility will have been employed to install optimized parameter values as initial parameter values in the "parameter data" section of the PEST control file.

As discussed above, PWTADJ2 has two options in performing observation weights adjustment. If the "ng" option for the *use_groups* control variable is supplied in its command line, then the adjustment factor by which PWTADJ2 multiplies all observation weights is the same for all observation groups. However if the "g" option is selected, different adjustment factors are computed for different groups. In both cases the total objective function after weights adjustment will be equal to $n - p$, where p is selected through the final variable on the command line; if this variable is omitted, p is assumed to be zero. If the "g" option is selected then, as stated above, PWTADJ2 adjusts weights such that the contribution made to the objective function by each group individually is equal to $n_g(n-p)/n$ where n_g is the number of non-zero-weighted observations in a group, and n is the total number of non-zero-weighted observations in the entire PEST input dataset.

If, for any observation group, a covariance matrix is used instead of observation weights, PWTADJ2 does not alter that covariance matrix. Rather it writes to the screen the factor by which all elements of that matrix must be multiplied to achieve the objective function targets discussed above. If the covariance matrix is supplied in matrix file format, multiplication of this covariance matrix by a scalar can be implemented using the MATSMUL utility.

The following aspects of PWTADJ2's operations should be carefully noted.

1. If the PEST control file nominated on the PWTADJ2 command line instructs PEST to run in regularisation mode, PWTADJ2 does not adjust weights within any observation group whose name begins with "regul". Thus only the measurement component of the total objective function is altered and not the regularisation component.
2. If the PEST control file nominated on the PWTADJ2 command line instructs PEST to run in predictive analysis mode, PWTADJ2 does not adjust the weight assigned to the single member of the observation group "predict", irrespective of the setting of the PREDNOISE variable. Thus if the calibration process is informative of predictive noise, alterations to the weight assigned to the "prediction observation" through which this noise level is conveyed to PEST must be made by the user.
3. Irrespective of the NOPTMAX setting in the PEST control file read by PWTADJ2, this variable is set to 50 in the PEST control file written by PWTADJ2. (It has been found from experience that a user can very easily forget to alter this from its useful value of zero in the control file supplied to PWTADJ2, and not realize this until he/she returns to a computer expecting to find a completed PEST run, only to find a single completed model run.)

3.12 INFSTAT

Note. See also INFSTAT1 documented later in this manual.

3.12.1 General

The INFSTAT utility computes a number of observation influence statistics, viz. observation leverage, Cook's D, an influence statistic proposed by Hadi (1992), and DFBETAS. For a complete description of these, see Yager (1998), Hadi(1992), and references cited therein.

Suppose that an overdetermined parameter estimation problem involving n observations and

m parameters is defined by the equation:-

$$\mathbf{Xp} = \mathbf{h} + \boldsymbol{\varepsilon} \tag{3.1}$$

where the $n \times m$ matrix represents the linearised action of the model, and the covariance matrix of measurement noise is given by C(ε). Suppose that a measurement weight matrix **Q** is employed in estimation of parameters, such that:

$$C(\boldsymbol{\varepsilon}) = \sigma^2 \mathbf{Q}^{-1} \tag{3.2}$$

In most cases **Q** will be a diagonal matrix comprised simply of observation (and prior information) weights.

Define model-to-measurement residuals **r** as:-

$$\mathbf{r} = \underline{\mathbf{h}} - X\underline{\mathbf{p}} \tag{3.3}$$

where $\underline{\mathbf{p}}$ are optimised parameters, computed using the formula:-

$$\underline{\mathbf{p}} = (\mathbf{X}^t\mathbf{Q}\mathbf{X})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{h} \tag{3.4}$$

and $\underline{\mathbf{h}}$ are model outputs to which there are corresponding field measurements computed on the basis of optimised parameters $\underline{\mathbf{p}}$.

Define:-

$$\mathbf{Z} = \mathbf{Q}^{1/2}\mathbf{X} \tag{3.5}$$

$$\mathbf{f} = \mathbf{Q}^{1/2}\mathbf{r} \tag{3.6}$$

$$\mathbf{H} = \mathbf{Z}(\mathbf{Z}^t\mathbf{Z})^{-1}\mathbf{Z}^t \tag{3.7}$$

Let h_{ii} be the *i*'th diagonal element of the "hat matrix" **H**. This is denoted as the "leverage" of observation *i*. It can be shown that:-

$$\frac{\mathrm{var}(\underline{h}_i)}{\mathrm{var}(f_i)} = \frac{h_{ii}}{1-h_{ii}} \tag{3.8}$$

Cook's D_i, a measure of the influence of observation *i*, is computed as:-

$$D_i = \frac{1}{m} v_i^2 \frac{h_{ii}}{1-h_{ii}} \tag{3.9}$$

where v_i is the weighted studentized residual corresponding to observation *i*, computed as:-

$$v_i = \frac{f_i}{s\sqrt{1-h_{ii}}} \tag{3.10}$$

s^2 is an estimate of the error variance σ^2 given by:-

$$s^2 = \Phi/(n-m) \tag{3.11}$$

and Φ is the minimized objective function given by:-

$$\Phi = \mathbf{r}^t\mathbf{Q}\mathbf{r} \tag{3.12}$$

See Yager (1998), and Cook and Weisberg (1982) for a full explanation of the significance of these statistics.

Hadi (1982) defines a similar influence statistic as:

$$H_i^2 = \frac{m}{1-h_{ii}} \frac{f_i^2}{\left(f^t f - f_i^2\right)} + \frac{h_{ii}}{1-h_{ii}} \tag{3.13}$$

$DFBETAS_{ij}$ is a measure of the influence of observation *i* on regression parameter *j*. It is calculated as:-

$$\beta_{ij} = \frac{c_{ji}}{\sqrt{\sum_{k=1}^{n} c_{jk}^2}} \frac{f_i}{s(i)(1-h_{ii})} \tag{3.14}$$

where c_{ji} is an entry of the matrix product $\mathbf{C} = (\mathbf{Z}^t\mathbf{Z})^{-1}\mathbf{Z}^t$. $s(i)$ is given as follows (Belsley et al, 1980):-

$$s^2(i) = \frac{1}{(n-m-1)}\left[(n-m)s^2 - \frac{f_i^2}{1-h_{ii}}\right] \tag{3.15}$$

3.12.2 Using INFSTAT

INFSTAT is run using the command:-

```
INFSTAT case outfile [nsig]
```

where *case* is the filename base of a PEST control file pertaining to a completed PEST run, and *outfile* is the name of a file to which INFSTAT will write tabulated leverage, Cook's D, Hadi, and DFBETAS statistics. Optionally, for purposes of testing the effect on calculated statistics of limited precision sensitivities, the number of significant figures employed for representation of elements of the Jacobian matrix (i.e. the **X** matrix in the above equations) can be limited to *nsig* by the user. If omitted (which would normally be the case), then elements of the **X** matrix are represented with the precision with which they are read.

INFSTAT reads the PEST control file *case.pst*, the Jacobian matrix file *case.jco*, and the residuals file *case.res* from the completed PEST run. Where the previous PEST run was in regularisation mode, INFSTAT also reads the "resolution data file" *case.rdf* in order to obtain optimised regularisation weights. It calculates influence statistics employing the formulas presented above, and records these statistics in its nominated output file.

The following should be noted.

1. INFSTAT will cease execution with an appropriate error message if the previous PEST run was undertaken in predictive analysis mode.
2. In any calculation that involves the number of observations and prior information equations *n* featured in the parameter estimation process, the count of *n* does not include any observation and prior information equation with a weight of zero.
3. As noted in the PEST manual, parameters may have been improved slightly on the final optimisation run after calculation of the latest set of parameter sensitivities which occupy the Jacobian matrix file. The PARREP utility can be employed to build a new PEST control file with optimised parameters supplied as initial parameter values. If NOPTMAX is set to -1 in this file, PEST will calculate derivatives on the basis of these parameters, record these derivatives, as well as other statistics, and then cease execution. The user may wish to employ central derivatives for extra precision

if this is done. Note however that this procedure is inappropriate if PEST was run in regularisation mode as optimised regularisation weight factors are not transferred to the new PEST control file by PARREP.

4. In recording observation names to its output file, an underscore is added to those observation names that belong to a group for which a covariance matrix is supplied instead of observation weights. In this case, the "observations" for which influence statistics are calculated are not really the nominated observations at all, but linear combinations **j** of the original observations **h** defined as:-

   $$\mathbf{j} = \mathbf{C}^{-1/2}\mathbf{h} \qquad (4.44)$$

   where **C** is the user-supplied observation covariance matrix for the pertinent observation group.

5. Where PEST is run in regularisation mode, calculation of influence statistics includes the effects of regularisation prior information and regularisation observations. However, due to the fact that weights for these observations and prior information equations are calculated by PEST on the basis of a user-supplied target objective function instead of an optimised objective function, and do not reflect the uncertainty of these observations and prior information, Cook's D and DFBETAS statistics do not have the same meaning in this context. In fact, it is difficult to know just what meaning they have when PEST is run in regularisation mode. Nevertheless, they are still calculated and left to the user to interpret.

6. Where singular value decomposition is employed for solution of the inverse problem, INFSTAT cannot compute influence statistics, for the problem is underdetermined unless "the solution" is restricted to the same subspace of parameter space as that in which the previous inversion exercise was carried out (as determined through the singular value decomposition process). INFSTAT does not undertake SVD in order to define this subspace; rather it ceases execution with an appropriate error message.

7. Where SVD-assist was implemented for solution of the inverse problem, INFSTAT will indeed compute influence statistics. However the user should bear in mind that these pertain to the inverse problem as posed in terms of super parameters rather than base parameters; this is apparent from an inspection of the DFBETAS table produced by INFSTAT. To ascertain the base parameter composition of each super parameter, use the PCLC2MAT utility.

8. As well as influence and DFBETAS statistics, INFSTAT presents a summary of important observations (and parameters which they influence), "importance" being established on the magnitude of these statistics with respect to thresholds as outlined in the references below (and listed in the various printed summaries). It should be noted that the establishment of threshold is somewhat arbitrary (Hadis 1992), and in the case of the Hadis statistic, the relative influence of observations can have a large effect because the calculation of the threshold uses the mean and standard deviation of the influence values. Thus, if one or more observations are orders of magnitude more influential than the others, the cutoff might remove observations that are influential (Hadis, 1992). In addition to observations greater than the Hadis threshold, INFSTAT provides the user with a report of influence that lists the 90-100th percentile and 80-90th percentile for comparison.

9. If, for some reason, a DFBETAS statistic for a particular parameter/observation

cannot be calculated (as can sometimes occur because of a zero-valued denominator in equation 3.15), a value of -1.0E35 is recorded in the pertinent location on the INFSTAT output file. In many cases the zero-valued denominator is indicative of high influence, so these observations are listed as having DFBETAS influence. However, the value of DFBETA reported for these observations is not representative.

3.12.3 References

Belsley, D.A, Kuh, E. and Welsch, R.E., 1980. Regression Diagnostics: Identifying Influential Data and Source of Collinearity. John Wiley, New York.

Cook, R.D. and Weisberg, S., 1982. Residuals and Influence in Regression. *Monogr. Stat. Appl. Probability*, vol 18, Chapman and Hall, New York, 1982.

Hadi, A.S., 1992. A new measure of overall potential influence in linear regression. Computational Statistics and Data Analysis, 14, 1-27.

Yager, R.M., 1998. Detecting influential observations in nonlinear regression modelling. Water Resourc. Res., 34:7, 1623-1633.

3.13 WTSENOUT

3.13.1 General

"WTSENOUT" stands for "WeigTed SENsitivity and model OUTputs". WTSENOUT undertakes the following tasks.

1. It reads a PEST control file and corresponding JCO and RES files (these being the binary Jacobian file and the residuals file respectively produced as an outcome of running PEST).
2. It calculates $\mathbf{Q}^{1/2}\mathbf{X}$ where **Q** is the weight matrix and **X** is the Jacobian matrix for the current problem. As is explained elsewhere, sensitivities in this matrix pertain to only adjustable parameters; furthermore they include the effects of logarithmic transformation and of any parameters that may be tied to them.
3. It computes $\mathbf{Q}^{1/2}\mathbf{o}$ where **o** is the model output vector corresponding to best-fit parameters.
4. It writes $\mathbf{Q}^{1/2}\mathbf{X}$ to a binary JCO file (from which an ASCII version can be obtained using the JACWRIT utility).
5. It writes the $\mathbf{Q}^{1/2}\mathbf{o}$ vector to an ASCII file in PEST matrix file format (see elsewhere in this addendum for definition of this format).

The following features of WTSENOUT should be noted:-

1. WTSENOUT will cease execution with an appropriate error message if the PEST control file does not inform PEST to run in parameter estimation mode; thus an error condition will arise if PEST has been run in regularisation or predictive analysis modes. (If the previous PEST run did, in fact, employ one of these modes, it is a simple matter to create a new PEST control file in the new mode, use PARREP to supply it with previously optimised parameters as initial parameters, and employ the JCO2JCO utility to create a corresponding Jacobian matrix file. PEST can then be run with NOPTMAX set to -1 or -2, and with the "/i" switch employed on the command line, thus using this JCO file instead of re-computing sensitivities. In this way a PEST

input/output dataset for use with WTSENOUT can be constructed without the need to re-compute parameter sensitivities.)

2. It is possible that the best set of parameters, and the model outputs corresponding to these as found in the residuals file, will have been calculated on the very last PEST upgrade attempt, immediately prior to cessation of PEST execution. The Jacobian matrix stored in the JCO file will then corredpond to the iteration just prior to this, and hence will not correspond exactly to the optimised parameter set. If this is the case, and if the user would like exact correspondence between the JCO file and the set of model outputs calculated on the basis of best-fit parameters, a new PEST control file can be created containing optimised parameters using the PARREP utility. PEST can then be run with NOPTMAX set to -1 or to -2 to compute the pertinent Jacobian matrix.
3. WTSENOUT will accommodate the provision of covariance matrices for one or more observation groups in the PEST control file.

3.13.2 Running WTSENOUT

WTSENOUT is run using the command:-

```
wtsenout pestfile matfile jcofile
```

where

pestfile is the name of a PEST control file employed as the basis for a previous PEST run,

matfile is the name of the weighted model matrix output file (i.e. the file which will contain $\mathbf{Q}^{1/2}\mathbf{o}$), and

jcofile is the name of the output weighted sensitivity file (i.e the binary file which will contain $\mathbf{Q}^{1/2}\mathbf{X}$).

PEST reads the nominated PEST control file and the JCO and RES files (i.e the Jacobian matrix and residuals files) associated with it. It obtaines the names of these latter two files by affixing the pertinent extensions to the same filename base as that supplied for the PEST control file. (The user should ensure that the new JCO file written by WTSENOUT has a different name from this.)

matfile and *jcofile* are WTSENOUT output files. The first is a matrix file, which will be used for recording the $\mathbf{Q}^{1/2}\mathbf{o}$ vector. The second is a binary Jacobian file; this must have an extension of *".jco"*. An ASCII version of this file can be produced using the JACWRIT utility.

3.14 ADDREG1

3.14.1 General

ADDREG1 adds a simple set of regularisation prior information equations to a PEST control file. An equation is added for each adjustable (i.e. non-tied and non-fixed) parameter cited in the file. In each of these equations the parameter (or its log, depending on its transformation status) is assigned a value equal to its initial value (or the log of its initial value). Thus it is assumed (as is good practice in undertaking regularised inversion) that parameter initial values are preferred values.

All prior information equations are assigned a weight of 1.0. Each is provided with a name which is the same as the name of the parameter which it cites.

Collectively, the prior information equations added to the PEST control file by ADDREG1 are designed for use as a Tikhonov regularisation device. Hence each such equation is assigned to a regularisation group whose name begins with "regul". ADDREG1 provides the second part of this name with the first six letters of the parameter group to which the pertinent parameter belongs. With parameter reglarisation prior information equations thus assigned to different regularisation groups, the IREGADJ variable in the "regularisation" section of the PEST control file should be set to 1 to allow PEST to vary regularisation weights between groups, thus (hopefully) complimenting the information content of the calibration dataset as it pertains to parameters belonging to each parameter group. If such division of regularisation prior information equations into different groups is ether too restrictive or is insufficiently reflective of the different roles played by different parameter types, the user can alter the groups to which parameters are assigned in the original PEST control file and re-run ADDREG1, or simply manually edit the ADDREG1-generated PEST control file. Note that the names of the new regularisation groups are added to the "observation groups" section of the new PEST control file; hence the ADDREG1-generated PEST control file should receive the approval of PESTCHEK.

3.14.2 Using ADDREG1

ADDREG1 is run using the command:-

```
addreg1 pestinfile pestoutfile
```

where *pestinfile* is the name of an existing PEST control file and *pestoutfile* is the name of a PEST control file which ADDREG1 must write.

Irrespective of whether the first PEST control file requires PEST to run in "estimation", "predictive analysis" or "regularisation" mode, ADDREG1 sets the PESTMODE control variable to "regularisation" in the file which it writes. It also writes a "regularisation" section to the new PEST control file. If a "regularisation" section is present in the old PEST control file, values for PHIMLIM and PHIMACCEPT are transferred from the old file to the new one. If not, ADDREG1 assigns PHIMLIM a value of 1.0 and PHIMACCEPT a value of 1.05 in the new PEST control file; these may warrant alteration by the user. However a value for the FRACPHIM variable is recorded in the new PEST control file, this value being 0.1. Thus, during any iteration of the parameter estimation process, the target objective function "seen" by PEST is 0.1 times its value at the beginning of that optimisation iteration. Also, ADDREG1 assigns IREGADJ a value of 1.

3.15 RANDPAR

3.15.1 General

RANDPAR generates model parameter value realisations using a random number generator. The parameters for which it must generate realisations are read from an existing PEST control file. The RANDPAR-generated random numbers are inserted into a series of PEST parameter value files which complement the original PEST control file. The TEMPCHEK utility can then be used to insert these parameter sets into model input files. Alternatively, the PARREP utility can be employed to insert them into a series of PEST control files; if PEST is then run with NOPTMAX set to zero in each of these files, the objective function corresponding to each set of parameter values can be calculated.

Two options are provided for the distribution type through which the random parameter values are generated. The first option is to employ a uniform distribution. In this case the upper and lower end of the parameter range is taken as the upper and lower bounds of respective parameters as recorded in the PEST control file. If a parameter is log-transformed, then the log of the parameter is assumed to possess a uniform distribution, rather than the parameter itself. No correlation is permitted between parameter values if a uniform distribution is specified.

Where a normal distribution is specified, parameter variances/covariances must be supplied through a parameter uncertainty file (see elsewhere in this manual for a description of this file type). If a covariance matrix with non-zero off-diagonal elements is cited in this file, then correlation can exist between (some or all) parameters. It is important to note, that if a parameter is log-transformed in the PEST control file read by RANDPAR, then it is assumed that uncertainties and correlations supplied in the parameter uncertainty file pertain to the logs of the respective parameters rather than to the native parameters.

Random values are not generated for parameters which are fixed in the PEST control file read by RANDPAR. Nor are they generated for tied parameters; instead the values assigned to tied parameters are such that the ratio of initial values between tied parameters and those to which they are linked as supplied in the original PEST control file is preserved.

Parameter realisations can be filtered such that those that do not respect certain user-supplied ordering relationships as expressed in a "parameter ordering file" are rejected, to be replaced by realisations that do, in fact, respect these relationships.

3.15.2 Using RANDPAR

RANDPAR data inputs are supplied through user responses to prompts, rather than through the command line.

RANDPAR commences execution by prompting for the name of the PEST control file which it must read:-

```
Enter name of existing PEST control file:
```

in response to which the name of an existing PEST control file should be supplied. Next it asks:-

```
Use (log)normal or (log)uniform distrib for param generation? [n/u]:
```

Respond with "n" or "u" as appropriate. Note that the choice of lognormal/normal or loguniform/uniform distribution is made on the basis of the parameter transformation status as recorded in the PEST control file. Note also that random numbers are not generated for fixed or tied parameters. The former retain their fixed value in all realisations; the latter are assigned values that preserve the initial value ratios as defined in the PEST control file.

If any parameters are tied, RANDPAR asks:-

```
Respect parameter ranges (tied parameters)? [y/n]:
```

This question is necessary because in preserving the ratios between tied and parent parameter values, the former may transgress their bounds as the latter are assigned random values between those bounds. If the answer to the above prompt is "n", then the fact that tied parameters may transgress their bounds is ignored. If it is "y", then tied parameters are clipped at their bounds, thereby destroying the tied parameter – parent parameter ratio decreed in the PEST control file.

If a normal distribution is requested, RANDPAR needs to acquire more information than it does for generation of parameter sets according to a uniform distribution. It asks:-

```
Compute means as existing param values or range midpoints? [e/m]:
```

The user is presented with two choices here. In generating random numbers which belong to a normal distribution a mean value is required. RANDPAR can accept the initial parameter value as supplied in the PEST control file as the mean value of each parameter. Alternatively it can compute the mean as midway between the lower and upper bound of each parameter as supplied in the PEST control file (or midway between the logs of the lower and upper bounds if a parameter is log-transformed).

Next (for a normal distribution only) RANDPAR asks:-

```
Respect parameter ranges? [y/n]:
```

or if any parameters are tied:-

```
Respect parameter ranges (parent parameters)? [y/n]:
```

If the response to the above prompt is "y", then if any parameter realisation lies below or above the upper bound of the parameter as recorded in the PEST control file, RANDPAR will assign the parameter a value equal to the parameter lower or upper bound respectively. Alternatively, if the answer is "n", bounds will be ignored in the generation of parameter values.

If any parameters are tied, RANDPAR next prompts (as it does for the uniform distribution option):-

```
Respect parameter ranges (tied parameters)? [y/n]:
```

The response to this question has the same effect as it does where parameters are assigned a uniform distribution, as discussed above.

If a normal distribution was specified for the generation of random realisations, the name of a parameter uncertainty file is next acquired. The format of this file is discussed in documentation for program RESWRIT. Within this file, uncertainties can be supplied on a parameter-by-parameter basis; alternatively one or a number of parameter covariance matrices can be supplied.

RANDPAR next asks (for both normal and uniform distributions):-

```
Enter name of parameter ordering file (<Enter> if none):
```

An example of a parameter ordering file follows:-

```
# parameter ordering file

h2 > h1
ro2 > ro1
ro10 > ro2
```

A Parameter Ordering File.

Each line of a parameter ordering file must cite the names of two parameters; these parameters can be adjustable, tied or fixed in the PEST control file. Parameter names must be separated by a ">" or a "<" character. Parameter realisations which do not satisfy the supplied relationships will be rejected. Thus, for any one parameter value file which it writes, RANDPAR will continue to generate realisations until a realisation is finally obtained for which all relationships expressed in the parameter ordering file are respected. However if,

after 1000 attempts, such a parameter set is not achieved, RANDPAR will cease execution with an appropriate error message. The user is then advised to set parameter ordering relationships less restrictively (or alter the variables governing the probability functions on which basis parameter realisations are generated), and then re-run RANDPAR.

RANDPAR's next prompt (irrespective of distribution type) is:-

```
Enter integer random number seed (<Enter> if default):
```

Supply a positive integer, or simply press the <Enter> key. Random number sequences are identical for the same random number seed; to generate different sets of parameter values, use different seeds.

Finally RANDPAR asks:-

```
Enter filename base for parameter value files:
How many of these files do you wish to generate?
```

Suppose that you supply a filename base of "*base*". Then parameter value files written by RANDPAR are named *base1.par, base2.par, base3.par* etc.

3.16 JCODIFF

JCODIFF subtracts the contents of one Jacobian matrix file from that of another. It can be useful when exploring the uncertainties or error variances pertaining to predictive *differences* rather than to predictions themselves. The former are often far smaller than the latter.

JCODIFF is run using the command:-

```
jcodiff jcofile1 jcofile2 jcofile3
```

where

jcofile1	is an existing Jacobian matrix file,
jcofile2	is another existing Jacobian matrix file, and
jcofile3	is a new Jacobian matrix file, written by JCODIFF, containing a Jacobian matrix which is the difference between the above two.

Note the following.

1. The first and second matrix files cited on the JCODIFF command line must contain the same number of rows and columns; they must also cite the same parameters/observations in the same order.
2. The matrix contained in the second Jacobian matrix file is subtracted from that contained in the first in forming the matrix written to the third Jacobian matrix file.

3.17 JCOCOMB

3.17.1 General

JCOCOMB reads one Jacobian matrix file (i.e. JCO file) and writes another. Observations pertaining to the second JCO file are user-specified linear combinations of those recorded in the first; hence JCOCOMB combines sensitivities in the same proportions.

Use of JCOCOMB requires that a PEST control file and complimentary JCO file be already in existence. A second PEST control file must also have been prepared. This must cite the same parameters in the same order as the first control file. Also, all parameters must have the

same tied/fixed/transformation status in the second PEST control file as they do in the first. However observations cited within the second PEST control file will be different. It is JCOCOMB's task to compute the sensitivities of these observations to all parameters.

3.17.2 Observation Combination File

The relationship between observations cited within the second PEST control file and those cited within the first PEST control file must be supplied by the user through an "observation combination file". An example of such a file follows.

```
* composite_observation "car1"
ar1  1.0
ar2  2.0
* composite_observation car2
"ar17" 0.5
ar18
ar16 0.5
* composite_observation car3
pi1 0.2
pi2 0.3
```

An observation combination file.

An observation combination file is subdivided into sections. Each such section must begin with a string of the type "***** composite_observation *observation_name*" where *observation_name* is the name of an observation cited in the second PEST control file. This can be surrounded by quotes if desired.

Each of the following lines within each section of the observation combination file should contain two entries. The first is the name of an observation cited in the first PEST control file (optionally surrounded by quotes). The second is the factor by which its sensitivities with respect to each parameter should be multiplied in forming the sensitivity of the composite observation to which that section of the observation combination file pertains. Each section of the observation combination file can have as many entries as desired. Sensitivities for each of the observations cited within that section (as read from the JCO file associated with the first PEST control file) are simply multiplied by the pertinent factor and summed (for each adjustable parameter) to compute the sensitivities of the composite observation to all such parameters. Composite observation sensitivities are then recorded in a JCO file that compliments the second PEST control file.

The following should be noted.

1. All observations cited in the second PEST control file should be cited in the observation combination file.
2. There is no requirement that all observations cited in the first PEST control file be cited in the observation combination file.
3. An observation from the first PEST control file can be cited more than once in the observation combination file; that is, it can contribute to more than one composite observation.

3.17.3 Using JCOCOMB

JCOCOMB is run using the command:

```
jcocomb casename1 casename2 combfile
```

where

casename1 is the filename base of the first PEST control file;

casename2 is the filename base of the second PEST control file; and

combfile is the name of an observation combination file.

JCOCOMB reads the first PEST control file and corresponding JCO file. It then reads the second PEST control file and then the observation combination file. Finally it writes a JCO file corresponding to the second PEST control file.

Note the following:-

1. JCOCOMB only reads the "control data", "parameter data" and "observation data" sections of the second PEST control file. There is thus no requirement for instruction files to be cited within this file which inform PEST how model outputs corresponding to these observations are to be read from model output files. So while this file may not be useable for a parameter estimation run, it is nevertheless useable by utilities such as the PREDVAR* and PREDUNC* suites which also ignore the instruction and template files cited within a PEST control file.
2. JCOCOMB makes no alterations to the second PEST control file. Also, the values assigned to these observations, and the weights that are assigned to them, are ignored by JCOCOMB.
3. The first PEST control file may cite items of prior information. Composite observations listed in the observation combination file may indeed cite the names of these prior information equations as "observation names" within respective sections of that file. However no prior information must be cited in the second PEST control file.

JCOCOMB can be useful prior to PREDVAR* and PREDUNC* processing, where predictive errors and uncertainties are computed for averaged quantities and compared with those pertaining to individual quantities. In many circumstances the error and uncertainty variances associated with averaged quantities are significantly less than for predictions pertaining to fine system detail. When used in this manner, the composite observations contained within the second PEST control file will actually be model predictions made under user-specified model input circumstances. Sensitivities written to the second JCO file by JCOCOMB will then be extracted for the use of the PREDVAR* and PREDUNC* by the JROW2VEC utility prior to PREDVAR* and PREDUNC* processing.

3.18 PNULPAR

3.18.1 General

PNULPAR compliments the use of RANDPAR in undertaking Monte-Carlo analysis. However, while RANDPAR-generated fields can be employed in Monte Carlo analysis that does not enforce calibration constraints, PNULPAR can be employed in Monte-Carlo analysis that does.

PNULPAR reads a set of parameter value files generated by RANDPAR. Supposedly, the sets of random parameters contained within these files have been generated on the basis of a PEST control file (referred to herein as a "pre-calibration PEST control file") in which initial parameter values are expected (or average) parameter values within an area of interest, or at a

particular study site. (Note that the expected value can be different for each parameter.)

Use of PNULPAR is also predicated on the assumption that another PEST control file exists, similar to the first except for the fact that it contains calibrated parameter values as initial parameter values (this is referred to herein as the "post-calibration PEST control file"). Such a file can easily be produced using the PARREP utility following a PEST calibration run. A corresponding JCO file must exist; this may have been produced by running PEST on the basis of the post-calibration PEST control file with NOPTMAX set to -1 or -2 in that file. It could also have been produced (without the necessity to undertake any model runs) using the JCO2JCO utility (supplying the names of the first and second PEST control files after the calibration process is complete), or simply by copying the JCO file pertaining to the first PEST control file (computed on the basis of the calibrated parameter set achieved at the end of the previous PEST run) to a JCO file appropriately named to complement the second PEST control file (this having the same filename base as the post-calibration PEST control file, but possessing an extension of "*.jco*").

PNULPAR reads the sets of parameter value files produced by RANDPAR, writing a new set of parameter value files in which parameter values are modified from those occurring in the RANDPAR-generated files. In generating these new parameter value files PNULPAR undertakes the following operations.

1. It undertakes singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$ where **Q** is the observation weight matrix and **X** is the Jacobian matrix pertaining to the post-calibration PEST control file. (Because of the similarity of the two, **Q** thus also pertains to the pre-calibration PEST control file.) It asks the user for the dimensionality of the calibration solution space. This is the number of dimensions spanned by those columns of the **V** matrix corresponding to significantly non-zero singular values, where singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$ is described by the following formula:-

   $$\mathbf{Q}^{1/2}\mathbf{X} = \mathbf{USV}^t$$

   (In the following it is assumed that $\mathbf{V} = [\mathbf{V}_1\ \mathbf{V}_2]$ where $\mathbf{V}_2$ is an orthogonal matrix whose columns span the calibration null space.)

2. Suppose that the elements of the vector **p** comprise the set of parameter values contained within a particular RANDPAR-produced parameter value file. Let $\underline{\mathbf{p}}$ describe the set of parameters contained (as initial parameter values) within the post-calibration PEST control file read by PNULPAR; thus $\underline{\mathbf{p}}$ calibrates the model. PNULPAR next computes $\mathbf{p}\text{-}\underline{\mathbf{p}}$ (taking parameter transformation status into account).

3. Components of parameter differences encapsulated in $\mathbf{p}\text{-}\underline{\mathbf{p}}$ that possess a non-zero projection onto the calibration solution space are then removed by computing a set of "projected parameter differences" $\mathbf{p}_d$ using the equation:-

   $$\mathbf{p}_d = \mathbf{V}_2\mathbf{V}_2^t(\mathbf{p}\text{-}\underline{\mathbf{p}})$$

4. A new set of parameter values is then produced by adding $\mathbf{p}_d$ to $\underline{\mathbf{p}}$ (contained in the post-calibration PEST control file). These values are written to a new parameter value file.

With the exception of singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix (which is only undertaken once), PNULPAR undertakes each of the above operations for every existing RANDPAR-generated parameter value file that it encounters, writing a new parameter value file in each case as an outcome of these calculations.

Note that the ratios of tied parameters to parent parameters are preserved by PNULPAR (unless bounds are encountered – see below). In fact if the ratio of any tied parameter to its parent parameter in the post-calibration PEST control file read by PNULPAR is different from the ratio of these same parameters in any RANDPAR-generated parameter value file, PNULPAR will cease execution with an appropriate error message. (This should cause no problems in normal RANDPAR operation, as RANDPAR preserves the ratio of tied to parent parameters as it generates random values for the latter.)

The handling of fixed parameters is different. If a fixed parameter has a different value in a parameter value file from that in the PEST control file, the value in the parameter value file prevails. This strategy has advantages in certain instances of groundwater model calibration where, for example, the PPSAMP utility is used instead of RANDPAR for generation of random parameter values. However the problem should never occur with RANDPAR as it respects the value of fixed parameters, and does not generate random values for them.

3.18.2 Using PNULPAR

PNULPAR commences execution with the prompt:-

```
Enter name of PEST control file:
```

Supply the name of a post-calibration PEST control file whose initial parameter values are in fact calibrated values from a previous PEST run (normally having been inserted into this file using the PARREP utility). A JCO file must exist, corresponding to this PEST control file. As stated above, this can be produced using JCO2JCO or the "copy" command. However following an SVD-assist run (after which best parameters are PARREPed into the post-calibration PEST control file from a BPA file whose filename base is the same as that of the base PEST control file on which SVD-assisted calibration was based), a post-calibration JCO file should be produced through running PEST on the basis of the post-calibration base PEST control file with NOPTMAX set to -1 or -2 in this file.

PNULPAR next prompts:-

```
Does PEST control file contain calibrated parameter values? [y/n]:
```

If you answer "n" to this question, PNULPAR will cease execution with some salient advice. Otherwise it prompts:-

```
Enter number of dimensions of calibration solution space:
```

The dimensions of the null space are *m* minus this number, where *m* is the number of adjustable parameters cited in the PEST control file. The dimensions of the calibration solution space will be equal to the number of singular values employed in SVD-based solution of the inverse problem, or the number of super parameters employed in SVD-assisted parameter estimation. Alternatively you can estimated it by undertaking singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix yourself using the MATSVD utility and counting the number of significantly non-zero singular values. To facilitate this process, PNULPAR stores the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix in PEST matrix file format if requested. It prompts:-

```
Would you like to store Q(1/2)X matrix in matrix file format? [y/n]:
```

and if your answer to this question is "y":-

```
Enter file for storage of matrix:
```

Next PNULPAR asks:-

```
Enter filename base of existing parameter value files:
```

Suppose that your answer to the above question is "*base*". Then PNULPAR will look for files *base1.par, base2.par* etc. When it encounters no more files in this sequence, it assumes that it has found them all. For each such parameter value file it writes a new parameter value file based on null-space-projected parameter differences as described above. The filename base of these new files must be supplied in response to the prompt:-

```
Enter filename base for new parameter value files:
```

An extension of "*.par*" is again assumed. The numeric suffix in each case is the same as that of the corresponding, existing, RANDPAR-produced, parameter value file.

Having been supplied with all of its input data requirements, PNULPAR undertakes the computations outlined above, outlining its progress to the screen. Then it ceases execution.

Note that PNULPAR will not accept a PEST control file in which PESTMODE is set to "prediction" or "regularisation". So if you wish to employ PNULPAR in the regularised inversion context (as is often the case), alter PESTMODE to "estimation" and remove all regularisation observations and prior information equations from the post-calibration PEST control file. Then either set NOPTMAX to -2 and run PEST to obtain a JCO file, or simply use the JCO2JCO utility. (As described in its documentation, JCO2JCO easily accommodates missing observations and/or prior information.) The reason for PNULPAR's refusal to read a PEST control file in which PESTMODE is set to "regularisation" is to avoid the potential for confusion and error; singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix (through which solution and null spaces are defined) should not include any regularisation information.

3.18.3 What to do Next

Parameter values recorded in parameter value files generated by PNULPAR can be PARREPed into a PEST control file. NOPTMAX can be set to zero in this file and the objective function computed in each case. If the model is linear, this objective function should be the same as that achieved during the previous calibration exercise (or almost the same, depending on how sharp is the singular value cut-off between parameter and solution spaces).

Where a model is nonlinear, the use of PNULPAR-generated parameters will probably not result in a calibrated model. In many cases, however, parameter values can be "warped" back into calibration with very little effort, often requiring only one PEST iteration. (If desired, set NOPTMAX to 1 to ensure that only one iteration takes place.) Furthermore, this process can be made even more inexpensive by employing sensitivities already present in the existing JCO file computed under calibration conditions (the same JCO file as that read by PNULPAR). This can be achieved by starting PEST with the "/i" switch and providing the name of the pertinent JCO file when prompted accordingly.

When undertaking a single-iteration PEST run under these conditions, the following may help.

1. Set the BROYDEN update parameter JACUPDATE to a high number. Thus on the second and further attempts to upgrade parameter values during this single iteration, PEST will have improved the Jacobian matrix on whose basis these upgrades are computed, this resulting (hopefully) in a lower objective function.
2. Set the Marquardt lambda to a reasonably low number – for example 1.0.
3. Set the PHIRATSUF control variable very low (for example 0.001). Thus PEST will be forced to test a number of different Marquardt lambdas before the end of the

iteration. Hence an objective function can be found which is as low as it can possibly be based on current sensitivities. Also, on the second upgrade attempt, the benefits provided by the Broyden process will come into play.

Where SVD-assist is employed as a calibration device, model re-calibration following introduction of each new PNULPAR-generated parameter set can be achieved with even greater model run efficiency. Suppose that the base PEST control file for the original calibration run is named *base.pst* and that the corresponding JCO file (the file on whose basis super parameters are computed) is named *base.jco*. It is assumed that initial parameter values contained in *base.pst* are the expected values for these parameters based on knowledge of the study site existing prior to implementation of the calibration process. Suppose further that the super parameter PEST control file (built using SVDAPREP) is named *super.pst*. It is assumed that SVDA_SUPDERCALC is set to 1 in this file (this is the SVDAPREP default). Thus PEST undertakes no model runs for the purpose of super parameter derivatives calculation during its first iteration; rather super parameter derivatives are automatically computed on the basis of base parameter derivatives contained in *base.jco*.

After the calibration process has been completed, a file named *base.bpa* will have been written by PEST (which will have been run on the basis of *super.pst*), this containing optimised base parameter values. The following steps can then be undertaken to obtain a set of random parameter values which also calibrate the model. (This process can then be repeated as many times as desired in order to implement Monte-Carlo based model predictive error analysis for any number of desired predictions.)

1. Use PARREP to transfer calibrated parameter values contained in *base.bpa* to a new PEST control file named *base1.pst* based on *base.pst*. This can be done using the command:-

   ```
   parrep base.bpa base.pst base1.pst
   ```

2. Next set NOPTMAX to -2 in *base1.pst* and run PEST in order to compute base parameter sensitivities, these being stored in *base1.jco*.

3. Copy file *base1.pst* to *base2.pst*. Remove regularisation observations and prior information from this file and tell PEST to run in "estimation mode". (PNULPAR requires this.) Then use JCO2JCO to create *base2.jco* from *base1.jco* using the command:-

   ```
   jco2jco base1 base2
   ```

4. Use RANDPAR to generate a suite of parameter value files on the basis of *base.pst* and a relevant parameter uncertainty file.

5. Use PNULPAR to generate a set of null-space-projected parameter values based on the RANDPAR generated parameter values produced in step 4 above. Cite *base2.pst* as the pertinent PEST control file.

6. In the "svd assist" section of *super.pst*, alter *base.pst* to *base1.pst* and *base.jco* to *base1.jco*. Thus PEST is instructed to look for initial parameter values in *base1.pst* and for the Jacobian matrix on which to define super parameters in *base1.jco*.

7. In *super.pst*:-

   a. Set RLAMBDA1 to 1.0
   b. Set PHIRATSUF to 1e-3
   c. Set JACUPDATE to 999

d. Set RELPARMAX to 0.4
e. Set NOPTMAX to 1

8. For each PNULPAR-produced parameter value file, issue the command:-

```
parrep filen.par base.pst base1.pst
```

where "*filen.par*"is the name of the *n*'th parameter value file written by PNULPAR. Then run PEST using the command:-

```
pest super
```

Thus PEST undertakes one iteration of SVD-assisted parameter estimation from initial parameter values computed by PNULPAR. Furthermore because SVDA_SUPDERCALC is set to 1 in *super.pst*, no model runs are required for derivatives calculation.

Note that RELPARMAX is set to 0.4 in *super.pst* (the SVDAPREP default is 0.1) in order to allow PEST to provide maximum parameter improvement on the single iteration granted to it. (If problems arise, be prepared to set this to 0.3 or lower.) If Tikhonov regularisation is employed, it may also be a good idea to lower the target objective function slightly in order to allow PEST to "aim low", thus improving its chances of reaching this target in one iteration.

3.19 DERCOMB1

3.19.1 General

"DERCOMB" stands for "derivatives combination". The "1" suffix is included in the name of this utility in anticipation of the fact that it will probably not be the last utility written to perform a similar role.

DERCOMB1 is made for use in calibration contexts where a model provides its own derivatives. As explained in the PEST manual, these are read from an external derivatives file written by the model. This file can be of a compressed or uncompressed type.

Where a model run by PEST is comprised of different executable programs, each of which can compute derivatives of its own outputs with respect to some or all of the parameters involved in the current PEST run, then the derivatives file written by each of them will be incomplete. The role of DERCOMB1 is to build a complete derivatives file by combining such partial derivatives outputs.

DERCOMB1 must be provided with the names of two derivatives files. The first of these must be a standard PEST external derivatives file which is "complete" in the sense that its dimension are NOBS×NPAR where NPAR and NOBS are the numbers of parameters and observations respectively employed by PEST. Such a file may have been produced, for example, by the ASENPROC Groundwater Data Utility. Elements of the derivatives matrix that ASENPROC cannot compute are supplied with a dummy value of -1.11E33, this being PEST's expected value for Jacobian elements which have not been determined.

The second derivatives file must be in PEST matrix file format. A derivatives file in matrix format is written, for example, by the ZONE2VAR1 Groundwater Data Utility. Like the first file, this file must also contain derivatives of model outputs with respect to parameters as its elements. However, unlike the first file, it is not necessary that all parameters and observations cited in the PEST control file be cited in this file. Thus the matrix can have dimensions that are smaller than NOBS×NPAR. The names of the observations and

parameters to which the derivatives in this file pertain are provided as matrix row and column names respectively. Each one of these must correspond to an observation or parameter featured in the PEST control file for the current problem.

DERCOMB1 reads the first derivatives file, and then the second derivatives file. It verifies that the first has dimensions of NOBS×NPAR. (It does not check parameter and observation names in this file, as this file does not need to provide these; it is simply assumed that parameters and observations are represented in the same order as they are in the PEST control file for the current inversion problem.) DERCOMB1 then reads the matrix file. Every derivative represented in the matrix file then relaces the corresponding derivative represented in the external derivatives file. DERCOMB1 then writes a new external derivatives file with these modified derivatives.

3.19.2 Using DERCOMB1

DERCOMB1 is run using the command:

```
DERCOMB1 pestfile matfile derfile1 derfile2
```

where

pestfile is the name of a PEST control file,

matfile is the name of matrix file,

derfile1 is the name of an existing external derivatives file, and

derfile2 is the name of a new external derivatives file.

As stated above, the existing PEST external derivatives file must contain a matrix (represented in either full or compressed form) with dimensions of NOBS×NPAR, where NOBS is the number of observations cited in the PEST control file and NPAR is the number of parameters cited in this file. The matrix file must contain a matrix whose dimension are equal to or less than these; however all observations and parameters cited as row and column names in the matrix file must also be cited in the PEST control file (though not necessarily in the same order). DERCOMB1 overwrites derivatives in the existing external derivatives file with those in the matrix file. It then writes a new external derivatives file with the upgraded derivatives. If the original derivatives file is supplied in compressed format, then the new derivatives file written by DERCOMB1 will also employ compressed format.

DERCOMB1 will normally be run as part of a model batch or script file cited in the "derivatives command line" section of a PEST control file. It will run after the model components that produce the external derivatives file on the one hand, and the matrix file of derivatives on the other hand. Where there are many model executables, DERCOMB1 can be run multiple times as part of the one model.

3.20 JACTEST

3.20.1 General

JACTEST is used to rapidly test the integrity of derivatives calculated by PEST. As is stated in the PEST manual, success of the Gauss Marquardt Levenberg parameter estimation process, and particularly of regularised inversion involving many parameters, depends on the integrity with which derivatives of model outputs with respect to adjustable parameters are calculated. If numerical instability of the model presents an obstacle to the integrity of

derivatives calculations then this should be recognised. If derivatives' integrity is compromised, this does not necessarily mean that PEST cannot be employed with that model. However high-end PEST usage (particularly predictive uncertainty analysis and SVD-assisted parameter estimation involving a large number of super parameters) may be problematical.

3.20.2 Using JACTEST

JACTEST is run using the command:-

```
jactest pestfile parname n outfile [/p]
```

where:-

`pestfile`	is the name of a PEST control file,
`parname`	is the name of a parameter featured in this file,
`n`	is the number of increments to test,
`outfile`	is the name of the JACTEST output file, and
`/p`	is an optional parallelisation switch.

JACTEST reads a PEST control file. It then runs the model $n + 1$ times, where n is the number of increments supplied through the command line (it runs the model once without any increment). For each but the first of these model runs the value of a user-specified (through the command line) parameter is varied incrementally. The increment is the same as that which PEST would use to compute derivatives with respect to this parameter during the parameter estimation process, the control variables for which are provided in the "parameter groups" section of the PEST control file. Model outputs calculated on the basis of the incrementally-varied value of the specified parameter are stored internally. Parameter values are symmetrically disposed about the pertinent initial parameter value supplied in the PEST control file; that is, this parameter value is both incremented and decremented for the purpose of carrying out the requested model runs.

When all model runs are complete, JACTEST writes a table of model-generated observation values to a user-specified output file. The first row of this table contains values of the nominated parameter employed for each run. Subsequent rows contain the values of all model-generated observations calculated during each such model run (with the name of each observation as the first entry of each row). If the values of selected observations are then plotted against parameter values (this being easily achieved once the JACTEST-written table is imported into a spreadsheet), problems with derivative computation, if they exist, will be readily apparent.

Note the following.

1. Where a PEST control file specifies multiple command lines, JACTEST uses the first command to run the model.
2. JACTEST will cease execution with an error message if the user-nominated parameter is tied or fixed.
3. Where the user-specified parameter has other parameters tied to it, the tied parameters are varied in the same ratio as the parent parameter, in the same fashion as is done by PEST.
4. Parameter SCALEs and OFFSETs as provided in the PEST control file are respected

in the same manner as for PEST.

5. JACTEST will not transgress parameter bounds. If the addition of *n* times the derivative increment to a given parameter value causes that parameter to exceed its upper bound, JACTEST will subtract the increment instead of adding it to the base parameter value on successive model runs. If the lower bound is thereby transgressed, JACTEST will cease execution with an appropriate error message.
6. It may occur that the value of the user-specified parameter, as recorded in the JACTEST output file, is slightly different from its expected incremented value. This is a result of the fact that, like PEST, JACTEST adjusts its value slightly if the presence of a parameter space of limited width on one or more template files cited by PEST requires that parameter values be written with limited precision.
7. JACTEST does not list prior information values on its output file. (The derivatives of prior information outputs with respect to parameters always have integrity because they are simply the coefficients of respective parameters in the prior information equations.)

3.20.3 Using JACTEST in Parallel Mode

Model runs undertaken by JACTEST can be distributed across a network of computers by appending the "/p" switch to the end of its command line. In this case, JACTEST reads a run management file, the filename base of which is the same as that of the PEST control file, but whose extension is "*.rmf*". This file is described elsewhere in PEST documentation.

Operation of JACTEST in parallel mode is the same as that of PEST. Slaves (named PSLAVE) must be started in working directories on other machines (or maybe on the same machine if it has multiple processors). Communication between the master program (in this case JACTEST) and its slaves is through message files – the same message files as used by PEST. It is the slave, and not JACTEST, which runs the model in each case.

As for parallel PEST, slaves must be started before PEST. However, like parallel PEST, JACTEST will tolerate the late arrival of slaves.

When run in parallel mode, JACTEST writes two additional files. The first is a parallel run management file (with the same filename base as the PEST control file, but with an extension of "*.rmr*"); this documents the history of communication between JACTEST and its slaves. The second is a much shorter file which simply lists the number of runs completed at any time. This also has the same filename base as the PEST control file. However its extension is "*.rcr*" (for "run count record").

3.20.4 Stopping JACTEST

If the PSTOP or PSTOPST command is run in the JACTEST working directory, but from another window, JACTEST will cease execution. If run in serial mode, it will wait until the end of the current model run to do this. If run in parallel mode it will cease execution immediately, but cannot prevent the orphaned model run from reaching completion in the slave window. All slaves will automatically shut down if the PSTOPST command is used to terminate JACTEST execution; however they will not shut down if the PSTOP command is employed. Meanwhile JACWRIT writes all results that it has accumulated to the time of stoppage to its normal output file.

As for PEST, JACTEST execution can be paused and unpaused by issuing the PPAUSE and

UNPAUSE commands from another window which is opened in its working directory.

3.21 RDMULRES

3.21.1 General

RDMULRES stands for "ReaD MULtiple RESults". It is used to read identical data from many different model or PEST output files, the names of these files being distinguished from each other by an indicial integer. RDMULRES has many uses, two of which are now briefly discussed.

3.21.1 Monte Carlo Analysis

Monte Carlo analysis may be undertaken by running a model many different times using different sets of parameters on each occasion. Parameter values may have been generated by RANDPAR and may have been modified by PNULPAR. Alternatively they may have been generated by another program altogether.

Suppose that different sets of parameter values reside in parameter value files named *parval1.par*, *parval2.par* etc. Suppose that adjustable parameters reside in only one model input file, and that this file is named *model.in*, and that a template for this file is named *model.tpl*. Suppose further that the model output file to which results of interest are recorded is named *model.out*. Then a sequence of model output files named *model1.out, model2.out* etc containing model results calculated on the basis of parameter sets contained in *parval1.par*, *parval2.par* etc can be obtained using the following batch file if working on a PC. (A similar script file can be readily written for use on a UNIX platform.)

```
for /L %%i in (1,1,100) do (
tempchek parval%%i.par model.in model.tpl
model.exe
copy model.out model%%i.out)
```

A batch file in which a sequence of model runs is undertaken.

The above batch file runs the model executable program (in this case named *model.exe*) one hundred times. In the sequence "(1,1,100)" the first integer is the starting value, the second is the step size and the final integer is the finishing value. On each occasion the model output file is linked to the parameter set on whose basis it was recorded by integer index (represented by the *%%i* variable).

3.21.2 A Sequence of PEST Runs

PEST can be used to calibrate a model many different times using many different parameter sets as starting values. This can be useful in undertaking "null-space Monte Carlo" analysis where each of a sequence of PNULPAR-generated parameter sets must be "warped back into calibration". Suppose that parameter sets to use as initial values in subsequent PEST runs reside in parameter value files named *parval1par*, *parval2.par* etc. Suppose also that a suitable PEST file is named *case.pst* and that a copy of this named *case.pst.keep* has been made. Suppose also that, as explained in the documentation to PNULPAR, parameter estimation is actually carried out on super parameters, and that *case_svda.pst* is the PEST control file that is employed for this purpose (for which *case.pst* is the corresponding base PEST control file). Upon each occasion of cessation of *case_svda.pst* execution, optimised

base parameter values reside in *case.bpa*. However the run record file is named *case_svda.rec*.

The following batch file can be used to undertake multiple PEST runs on the basis of starting base parameter value housed in files *parval1.par*, *parval2.par* etc.

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

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

rem ############################################################
rem Do all the PEST runs.
rem ############################################################

for /L %%i in (1,1,100) do (
parrep parval%%i.par case.pst.keep case.pst
pest case_svda
find /I "optimised mea" case_svda.rec >> record.dat
copy case_svda.rec case_svda.rec.%%i
copy case.bpa case.bpa.%%i)
```

A batch file used for conducting a sequence of SVD-assisted parameter estimation runs.

After running the above batch file, subsequent run record files will be named *case_svda.rec.1*, *case_svda.rec.2*, etc. Respective optimised parameter value files will be named *case.bpa.1*, *case.bpa.2* etc. Note that a "global record file" named *record.dat* is also kept. After each PEST run, the value of the latest optimised measurement objective function is appended to this file. This allows the user to have in his/her possession a continuously-updated record of the success or otherwise of the sequence of PEST runs.

3.21.2 The RDMULRES Input File

RDMULRES requires a control file, an example of which is shown below.

```
# An example RDMULRES input control file.

* observations
initobj
finalobj
* instruction file
obj.ins
* model output file
case_svda*.rec
* integer list
1
4 - 10
12
15
16-100
* rdmulres output file
rdmulres.rec
```

An example RDMULRES control file.

The RDMULRES input control file is subdivided into a number of sections, each of which begins with a section header. Section headers are as shown in the above example; in each case their name is preceded by the "*" character followed by a space.

The "observations" section must contain a list of names for the numbers that must be read from model (or PEST) output files. These must be provided one to a line. As is the normal PEST protocol, these names must be 20 characters or less in length (and are case insensitive); names must be unique. Up to 100 such names can be provided. This limit is set in order to keep the RDMULRES output file at a tractable width.

The "instruction file" section of the RDMULRES control file must contain a single entry, this being the name of an instruction file. This instruction file must cite all observations named in the "observations" section of the RDMULRES input file (and no more). Instruction files must follow the normal PEST protocol. An instruction file suitable for reading the initial and optimised measurement objective function values from a PEST run record file is illustrated below.

```
pif $
$INITIAL CONDITIONS:$
$measurement$  $=$ !initobj!
$OPTIMISATION RESULTS$
$Optimised measurement$ $=$ !finalobj!
```

An instruction file which reads a PEST run record file.

The model output file which the instruction file is designed to read must be listed as the sole filename cited in the "model output file" section which immediately follows the "instruction file" section of the RDMULRES control file. RDMULRES requires that this filename contain at least one "*" character. In fact, many such output files are read by the same instruction file; the name of each is obtained by replacing the "*" character with an appropriate integer on each occasion.

The integers to employ in formulation of model output filenames in this fashion are listed in the "integer list" section of the RDMULRES control file. Either one or two integers can be listed on each line of this section. If two are listed, the second must be larger than the first, and must be separated from the first integer by a "-" (i.e. dash) character. (Negative numbers are not allowed.) In this case the "*" character is progressively relaced by all integers between and including the two nominated integers.

The final section of the RDMULRES control file is the "RDMULRES output file" section. Numbers read from model output files are recorded in tabular fashion in this file. See below.

If any line within a RDMULRES control file contains no characters, or begins with the "#" character, it is ignored.

3.21.3 Running RDMULRES

RDMULRES is run by typing the command:-

```
rdmulres infile
```

at the screen prompt, where *infile* is the name of its control file. If any errors are detected in this file, RDMULRES ceases execution with an appropriate error message. If not, it reads all requested model output files, recording to the screen its progress in this endeavour. If any errors are encountered in reading any such files, it ceases execution with an appropriate error message. However if a requested file is not present, it records this fact to the screen (and to its

output file) and proceeds to the reading of the next cited file.

An example of a RDMULRES output file is shown below. Observation names head all columns but the first in the table contained in this file; the first column contains integer index values used in forming the model output filename from which respective numbers in other columns are read.

```
index      initobj              finalobj
 1         9546.1300            415.1000
 2         1715.0300            196.3000
 3         File case_svda.rec.3 not found.
 4          545.0300            10.95000
```

Part of a RDMULRES output file.

A RDMULRES output file is easily imported into a spreadsheet or graphing program for further processing.

3.22 MULPARTAB

3.22.1 General

MULPARTAB stands for "MULtiple PARameter TABle". It can be used in the postprocessing of multiple PEST runs undertaken, for example, as part of a "null space Monte Carlo" analysis. In such an analysis, a model calibration process is repeated many different times using the same sets of parameters, but with different starting values or base parameter values (see documentation of the PNULPAR utility). The outcomes of such an analysis are a suite of run record files, and a suite of corresponding parameter value files. In most cases these files will have identical names except for the presence of an indicial integer at some location within each name.

Run record files can be processed using the RDMULRES utility. Such processing can reveal which calibration exercises resulted in parameter sets that gave rise to an objective function which can be considered low enough to call the model "calibrated". Those that do can then be assimilated into a single table using the MULPARTAB utility. This table can be imported into spreadsheet and/or graphing software for further processing and/or display.

3.22.2 Using MULPARTAB

MULPARTAB is run using the command:

```
mulpartab parfile listfile outfile
```

where

parfile is a generic parameter value file name,

listfile contains a list of integer indices, and

outfile is the name of a tabular output file.

The generic parameter value filename must contain at least one asterisk (i.e. "*" character). In forming the names of actual parameter value filenames, this asterisk is replaced by an integer in each case. The list of integers to employ in forming parameter value filenames is supplied in the integer list file (the second filename provided on the MULPARTAB command line). Thus if the first MULPARTAB command line argument is supplied as "*parval*.par*", then MULPARTAB will look for files named *parval1.par*, *parval2.par*, etc, if

the integers "1", "2" etc are supplied in the integer list file.

As the name suggests, an integer list file must contain a list of integers. An example of such a file follows.

```
# An integer list file

1
2-4
5
7-20
```

Part of an integer list file.

Lines beginning with "*" or "#" are ignored in an integer list file; so too are blank lines. Integers on other lines can be supplied individually, or as the beginning and end members of a sequence separated by a "-" (dash) character. Negative integers are not allowed.

A MULPARTAB output file is shown below. As is apparent, the file is comprised of multiple columns, the first of which contains parameter names. Parameter values as read from parameter value files follow these. Headers for these latter columns are integer indices comprising part of the filename from which parameters in the respective column were read.

```
                      1            4            6
 ro1             0.9988774     1.348253     1.089392
 ro2              1.040405     1.884527     1.277791
 ro3              1.256830     3.430308     1.930116
 ro4              1.771721     7.359193     3.583956
 ro5              2.611562     17.36130     6.978424
 ro6              3.000000     27.00000     9.000000
 ro7              2.139646     8.553685     4.517614
 ro8              1.265521     2.049192     1.677921
 ro9             0.9782547     1.074042     1.032021
 ro10            0.9753630    0.9830849    0.9765695
 h1              0.2500000    0.2500000    0.2500000
 h2              0.5000000    0.5000000    0.5000000
 h3               1.000000     1.000000     1.000000
 h4               2.000000     2.000000     2.000000
 h5               4.000000     4.000000     4.000000
 h6               8.000000     8.000000     8.000000
 h7               16.00000     16.00000     16.00000
 h8               32.00000     32.00000     32.00000
 h9               64.00000     64.00000     64.00000
```

A MULPARTAB output file.

The following should be noted.

1. As documented in the PEST manual, the first line of a parameter value file must contain the word "single" or "double" followed by the word "point" or "nopoint". If these are absent from a parameter value file which MULPARTAB is asked to read, it will terminate execution with an error message.
2. SCALE and OFFSET values are also provided in a parameter value file. MULPARTAB multiplies parameter values by their SCALE and adds their OFFSET before recording these values to its output file.
3. All parameter value files read by MULPARTAB must cite the same parameters. However parameters do not need to be listed in the same order in each file.
4. If an expected parameter value file is absent, MULPARTAB will notify the user of

this fact, and then proceed to read the next parameter value file. Values listed for missing parameters are replaced with the "---" string in the respective column of the MULPARTAB output file.

5. If a parameter value file is, in fact, found, and if an error is encountered in reading this file, MULPARTAB will cease execution after recording an appropriate message to the screen.

3.23 COMFILNME

COMFILNME stands for "COMpress FILe NaMEs". This simple utility can sometimes be useful in winnowing unwanted output files after many related model or PEST runs have been carried out, prior to undertaking another analysis step on the basis of the reduced file collection.

Suppose, for example, that many PEST runs have been undertaken as part of a null-space Monte Carlo exercise, and that this has resulted in a sequence of parameter value files named *pestcase1.bpa, pestcase 2.bpa, pestcase 3.bpa,* etc. Suppose that corresponding run record files are named *pestcase1.rec, pestcase2.rec, pestcase3.rec* etc. Use of RDMULRES on the latter set of files may reveal that a suitably low objective function may not have been achieved on all of these runs. (This may have occurred if, for example, the NOPTMAX variable for these runs was set to 1 so that PEST was allowed to undertake just one iteration in which super parameter derivatives are available "for free" – see documentation of PEST's SVDA functionality for details.)

Suppose now that a prediction is to be made using the model, and that this prediction is to be made using many different parameters sets which calibrate that model, but from which those which do not calibrate the model must be excluded. Hence non-compliant parameter value files need to be removed from the set of files generated during the previous set of PEST runs. This could be achieved by simply deleting these files. However there are certain advantages to retaining a sequential numbering system for parameter value files in future processing of this nature. COMFILNME allows such deletions to take place, with higher index filenames being assigned lower indices to "fill in deletion gaps".

COMFILNME is run using the command:

```
comflenme datfile listfile
```

where:-

datfile is a generic data or text file name, and

listfile contains a list of integer indices.

The filename supplied as the first COMFILNME command-line argument must contain at least one asterisk ("*") character. In forming actual filenames this character is replaced by a sequence of integers. This sequence is listed in the integer list file whose name is supplied as the second argument to COMFILNME. The format for an integer list file is provided in documentation to MULPARTAB.

When run in the above manner, COMFILNME reads each of the nominated data files – that is, each of the files whose name is formed by replacing the "*" character in *datfile* by the integers in the list file. It then generates a sequence of files in which the "*" character is replaced by "1", "2", "3" etc in sequence, with no gaps. In doing this, previous files are overwritten.

The following should be noted.

- Files represented by *datfile* must be ASCII (i.e. text) files, and be no more than 3000 characters in width.
- Integers must be supplied in the integer list file in ascending order.
- Suppose that a total of *N* integers are directly cited or implied (in expressions of the type *n1 – n2*) in the integer list file. At the end of a COMFILNME run, members of the *datfile* sequence with integer indices beyond *N* will still remain in the user's directory. It is the user's responsibility to delete these.
- It is a simple matter to relate new file names to old file names. New file indices are simply "1", "2", "3" etc, with these indices assigned to files sequentially in order of their representation in the integer list file supplied to COMFILNME as its second command line argument.

3.24 PARAMID

3.24.1 General

PARAMID stands for "PARAMeter IDentifiability analysis". It performs simple analysis of the contents of an "SVD file" written by PEST, listing the contributions made by adjustable parameters involved in the current parameter estimation process to the eigenvectors spanning the calibration solution space.

Use of PARAMID is based on the following assumptions.

1. A PEST run has just been carried out in which singular value decomposition was employed for solution of the inverse problem.
2. The EIGWRITE variable was set to 1 in the PEST control file, thus ensuring that the full eigenvector matrix was written to the SVD file generated by PEST under these circumstances.
3. The initial Marquardt lambda was set to zero (and NUMLAM set to 1).
4. At least one optimization iteration was carried out before termination of PEST execution.

Note that if model run times are long and a Jacobian matrix file already exists based on a PEST run using another solution strategy, there is no need to repeat the run simply to implement the SVD solution mechanism. Simply alter the PEST control file so that the above conditions are met (after maybe using the PARREP utility to place optimized parameter values into this file), and start PEST using the /i switch. It will then prompt the user for the name of an existing Jacobian matrix file rather than calculating the Jacobian matrix itself.

3.24.2 Using PARAMID

PARAMID is run using the command:-

```
PARAMID pestfile outfile
```

where *pestfile* is the name of an existing PEST control file (meeting the above requirements) and *outfile* is the name of the file to which PARAMID must write the outcomes of its analysis.

PARAMID reads the PEST control file and the SVD file written by PEST on its previous run. From the SVD file it determines the number of singular values before truncation, and therefore the dimensionality of the calibration solution space.

For each adjustable parameter listed in the PEST control file PARAMID determines the largest (absolute) contribution that this parameter makes to any of the solution space eigenvectors, as well as the smallest absolute contribution. It records these (as well as the corresponding eigenvector number) to its output file.

Don't forget that:-

1. All eigenvectors are normalized; hence the largest contribution that any parameter can make to an eigenvector is 1.0.
2. Eigenvectors are arranged in order of decreasing singular value. Therefore lower-numbered eigenvectors correspond to parameter combinations of greater identifiability.

The following are some conclusions that it may be possible to draw from an inspection of the PARAMID output file.

1. If the largest contribution that a parameter makes to a solution space eigenvector is very low, then the parameter is unidentifiable through the calibration process (and is probably insensitive).
2. If a parameter makes a moderate contribution to only one solution space eigenvector, then it may still be unidentifiable, for it may be highly correlated with another parameter.
3. If a parameter makes a significant contribution to a low-numbered eigenvector, then it can probably be well estimated on the basis of the current calibration dataset.

3.25 POSTJACTEST

JACTEST produces a large output file which can be used to assess the integrity of derivatives of all model outputs with respect to a parameter nominated on the JACTEST command line. The JACTEST output file is easily imported into a spreadsheet such as Microsoft EXCEL. Numbers along any row of this file can then be plotted. Ideally such a plot should reveal a straight line, or a smooth curve. If not, a potential problem in the finite-difference calculation of derivatives is indicated.

In many circumstances, the number of observations represented in a JACTEST output file is daunting. Some methodology for flagging rows of this file that are worthy of plotting is therefore required. POSTJACTEST peforms this role.

POSTJACTEST is run using the command:

```
postjactest jactestfile thresh outfile
```

where *jactestfile* is the name of a JACTEST output file, and *outfile* is the name of a file which POSTJACTEST will write. *Thresh* is a threshold value whose role is explained below. This must be set toa number greater than zero.

POSTJACTEST reads the JACTEST output file line by line. On any line it first computes the difference between the first and last numbers represented on the line (these corresponding to a particular model output value computed for a parameter incremented by zero, and then by several parameter increments respectively). Next it computes differences between successive

model output values. It then computes the difference between the maximum and minimum such model output difference as a fraction of the model output difference between the first and last entries on this line divided by the number of parameter increments giving rise to these outputs. This fractional difference is then recorded on POSTJACTEST's output file, together with the observation name to which this calculation pertains.

If the outer difference between model outputs, as well as differences between all neighbouring model outputs, on any particular line of the JACTEST output file are less than the threshold specified on the POSTJACTEST command line, then that line is skipped by POSTJACTEST; the pertinent observation is thus not represented on the POSTJACTEST output file.

The POSTJACTEST output file is easily imported into a spreadsheet such as EXCEL. It should then be ordered such that higher slope discrepancies are listed first. The user should then inspect lines of the JACTEST output file corresponding to observations listed early in this re-ordered table, graphing the numbers listed on each such line in the manner described above. If an observation does not have a very low sensitivity, and the plot deviates significantly from a straight line, this may indicate that poor finite difference derivatives are inhibiting good PEST performance.

3.26 MULJCOSEN

MULJCOSEN reads a sequence of Jacobian matrix files named *case.jco.N* where *N* represents a sequence of integers starting at 1. Such a sequence of Jacobian matrix files (i.e. "JCO files") is written by PEST if the JCOSAVEITN variable in the "control data" section of the PEST control file is supplied as "jcosaveitn".

MULJCOSEN is run using the command:

```
muljcosen casename obspar aname outfile
```

where *casename* is the filename base of a PEST control file, *obspar* must be supplied as either "obs" or "par", *aname* is the name of an observation or parameter featured in the nominated PEST control file, and *outfile* is a file to which MULJCOSEN must record its output.

MULJCOSEN computes the composite sensitivity of either an observation or parameter on the basis of each JCO file which it reads. The *obspar* command-line variable determines whether the item of interest is in fact an observation or parameter, while the *aname* command line variable identifies the particular observation or parameter for which such computation is required. The formulas through which composite observation and parameter sensitivies are computed are provided in the PEST manual. In both cases, these formulas involve observation weights. Hence as well as reading the sequence of Jacobian matrix files corresponding to the PEST case basename supplied on its command line, MULJCOSEN also reads the corresponding PEST control file, together with as any observation covariance matrices cited therein.

Composite sensitivities are written in tabular form to its output file. It is important to note that if PEST was run in regularization mode, regularization observations/prior information are ignored in computation of composite parameter sensitivities. If PEST was run in predictive analysis mode, the prediction is likewise ignored.

3.27 IDENTPAR

3.27.1 General

The purpose of the IDENTPAR utility is to assist in the computation of "parameter identifiability" indices.

Suppose that a large number of parameters is cited in a PEST control file. Suppose further that the calibration dataset does not provide sufficient information to allow each of these parameters to be individually estimated (as is often the case). But suppose you want to know which parameters are identifiable and which parameters are not, given available calibration data.

For the purpose of the present discussion, a parameter is said to be "identifiable" if its projection into the calibration solution space is roughly equal to its magnitude. Or, to put it another way, a parameter is identifiable if its projection onto the calibration null space is small. As has been extensively discussed elsewhere in PEST documentation, the solution and null spaces are orthogonal, and can be defined through undertaking singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix, where $\mathbf{X}$ is the Jacobian matrix computed on the basis of optimized parameters, or parameters which are considered to be of high likelihood if calibration has not yet taken place. $\mathbf{Q}$ is the weight matrix employed in the current calibration process (which is mostly – but not always – diagonal).

Singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$ yields $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$ matrices, defined as follows:-

$$\mathbf{Q}^{1/2}\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^t$$

where the matrix $\mathbf{V}$ is comprised of orthogonal unit vectors which collectively span parameter space. The matrix $\mathbf{S}$ is comprised of diagonal elements only, these arranged in order of decreasing size. These are the singular values of $\mathbf{Q}^{1/2}\mathbf{X}$. Let this matrix be divided into two submatrices $\mathbf{S}_1$ and $\mathbf{S}_2$, with the latter containing low and zero singular values. The corresponding columns of $\mathbf{V}$, these being contained in the $\mathbf{V}_2$ submatrix of $\mathbf{V}$, are the eigenvectors of $\mathbf{Q}^{1/2}\mathbf{X}$ which span the calibration null space. Columns of the $\mathbf{V}_1$ submatrix on the other hand (these corresponding to the $\mathbf{S}_1$ singular values), span the calibration solution space. Hence:-

$$\mathbf{U}\mathbf{S}\mathbf{V}^t = \mathbf{U}\begin{bmatrix}\mathbf{S}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{S}_2\end{bmatrix}\begin{bmatrix}\mathbf{V}_1^t\\ \mathbf{V}_2^t\end{bmatrix} = \mathbf{U}\mathbf{S}_1\mathbf{V}_1^t + \mathbf{U}\mathbf{S}_2\mathbf{V}_2^t$$

How do we know where the solution space ends and the null space begins? Moore and Doherty (2005) show that a useful place to draw the line is the point where the error variance of a prediction begins to rise as singular values and corresponding eigenvectors are moved from the null space to the solution space (and measurement noise is thus amplified in the estimation of parameter eigencomponents). However when calibrating a model we do not usually have predictive sensitivities at our disposal, and hence cannot determine the truncation point in this fashion.

Another means through which the truncation point can be determined is to examine the error variance of estimation of the eigenvectors comprising the columns of $\mathbf{V}$; these are of course available prior to, or during, the calibration process through singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$. These eigenvectors can in fact be employed as "predictions" when using the PREDVAR1 utility. Recall that use of PREDVAR1 allows calculation of the pre- and post-calibration (for various numbers of singular values) error variance of a prediction. There is no

reason why a "prediction" cannot be a linear combination of parameters comprising a column of the **V** matrix.

If PREDVAR1 is used to evaluate how the error variance of an "eigenvector prediction" changes as the number of singular values prior to truncation is progressively increased, it will be found (naturally enough), that no change in predictive error variance occurs until the singular value is reached corresponding to the eigenvector in question. Then the error variance may either fall or rise. In general, for eigenvectors corresponding to small singular values, the predictive error variance will fall as the eigenvector crosses from the null space to the solution space. For eigenvectors corresponding to large singular values however, the error variance will rise as the associated singular value is reached, this indicating that the pertinent eigencomponent should not be estimated at all; that is, the pertinent eigencomponent should reside in the null space. This then provides the mechanism through which we can determine where truncation should occur. (As such it also provides a mechanism for determination of the minimum number of super parameters to employ in an SVD-assisted parameter estimation process). In short, truncation should occur at that singular value where the error variance associated with estimation of its associated eigenvector rises rather than falls if it is included in the solution space.

Suppose that the number of singular values, and associated eigenvectors, to be included in the solution space is determined in the above (or some other) manner. We then have a means of evaluating the identifiability of a particular individual parameter. Suppose, for example, that the parameter in question is that corresponding to the second row of the **X** matrix. Thus it will pertain to the second row of the **V** matrix, and hence to each of the orthogonal eigenvectors comprising the columns of **V**. To the extent that elements of the $\mathbf{V}_1$ eigenvectors have a large second (in this case) element, this second parameter is featured strongly in the solution space; to the extent that these elements all approach zero, the parameter in question is not strongly featured in the solution space and is hence unidentifiable.

We define the identifiability of parameter *i* as:-

$$d_i = \Sigma(\mathbf{V}_{1i})^2$$

That is, the identifiability of parameter *i* is the sum of the squared i^{th} components of all eigenvectors spanning the calibration solution space. This is also equal to the square of the magnitude of the parameter unit vector after projection of that unit vector onto the solution subspace. Its highest possible value is 1.0; its lowest possible value is zero.

The "nonidentifiability" of parameter *i* is equal to $1.0 - d_i$. It is equal to the square of the magnitude of a unit parameter vector after projection of that unit vector onto the null subspace. A maximum value of 1.0 implies that the parameter is completely unidentifiable on the basis of the current calibration dataset.

It should be noted that if the identifiability of a parameter is 1.0, this does not mean that it can be estimated without any error. It only means that the null space contribution to the error of its estimation is zero; only measurement noise contributes to its estimation error.

As is discussed below, IDENTPAR provides the user with the option of undertaking singular value decomposition on either $\mathbf{Q}^{1/2}\mathbf{X}$ (as discussed above), or on $\mathbf{X}^t\mathbf{Q}\mathbf{X}$. Both of these processes yield an identical **V** matrix. Hence IDENTPAR outcomes computed on the basis of either choice should yield the same (or almost the same) results. However in some circumstances, decomposition of one of these is faster than that of the other.

3.27.2 Using IDENTPAR

IDENTPAR is run using the command:-

```
INDENTPAR pestfilebase N vecfilebase matfile identfile [/s or /r]
```

where

pestfilebase is the filename base of a PEST control file;

N is the number of singular values defining the calibration solution space;

vecfilebase is the filename base to which eigencomponent sensitivity vectors are to be written (supply this as "null" if these are not to be written);

matfile is the name of a matrix file to which the $\mathbf{V}_1$ matrix is to be written (supply this as "null" if this is not to be written);

identfile is the name of a file to which parameter identifiability data is to be written (supply this as "null" if this file is not to be written);

/s (optional) instructs IDENTPAR to undertake singular value decomposition on $\mathbf{X}^t\mathbf{Q}\mathbf{X}$, (the default), while

/r (optional) instructs IDENTPAR to undertake singular value decomposition on $\mathbf{Q}^{1/2}\mathbf{X}$.

IDENTPAR begins execution by reading the PEST control file corresponding to the user-supplied filename base. It reads parameter data and observation weights from this file (and any observation covariance matrices cited therein if these are employed for any observation groups instead of weights). It then reads the Jacobian matrix corresponding to this PEST control file. This file is assumed to have the same filename base as the PEST control file but an extension of "*.jco*"; it is thus assumed that this file exists and has been computed either on the basis of initial parameter values (set NOPTMAX to -1 or -2 to do this) or during a prior parameter estimation process.

IDENTPAR next forms the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix (or $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix depending on the user's choice of the /s or /r switch – where "s" stands for "square matrix" and "r" stands for "rectangular matrix") and carries out singular value decomposition of the chosen matrix. On the basis of *N* as supplied by the user on the IDENTPAR command line, it then records the following information. Note however, that any of the following three tasks can be disabled by providing a name of "null" for the pertinent filename base or filename in the IDENTPAR command line.

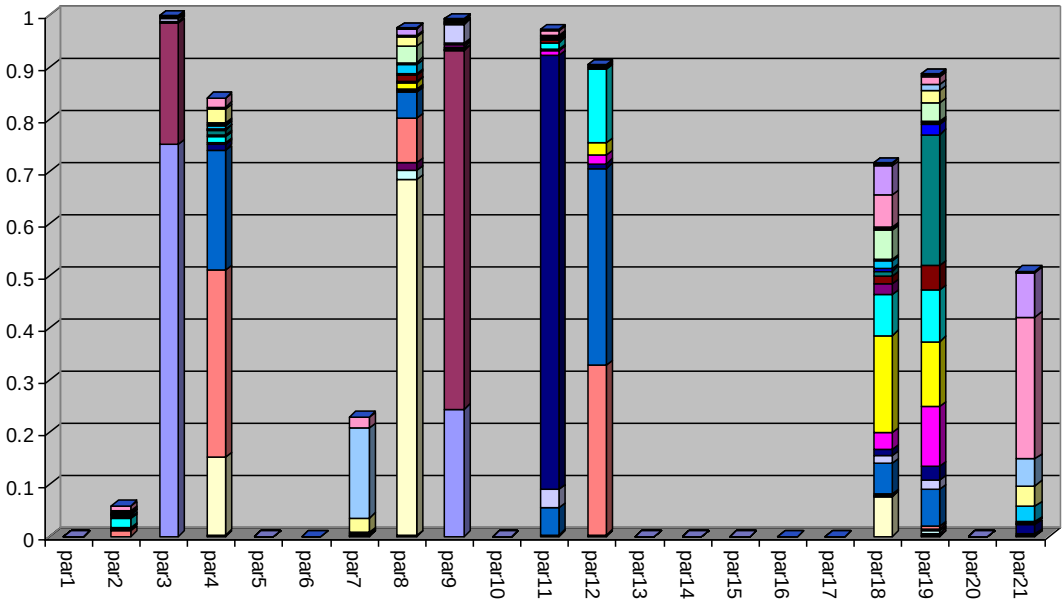
First IDENTPAR writes a series of "vector files" containing the columns of $\mathbf{V}_1$. Suppose that the user-supplied basename for these files is "*base*". Then the files will be named *base1.vec, base2.vec…..baseN.vec*. Each of these files can be used as a predictive sensitivity file by any of the PREDVAR or PREDUNC utilities. In particular, a sequence of these files can be used with PREDVAR1 in the manner discussed above for determining the value of *N* defining separation of parameter space into solution and null spaces. (If this is to be done, then *N* must obviously be supplied to IDENTPAR as larger than the anticipated value of the singular value cutoff; once the cutoff point is determined, IDENTPAR can then be re-run using the correct value of *N* to compute parameter identifiabilities.)

If *matfile* in the IDENTPAR command line is set to a value other than "null", IDENTPAR writes the columns of $\mathbf{V}_1$ to a single file in PEST matrix file format.

If *identfile* in the IDENTPAR command line is set to a value other than "null", IDENTPAR

computes the identifiability of each parameter and records it to the nominated file. However it also records the squares of the parameter components of all eigenvectors comprising the solution space (i.e. values for $(\mathbf{V}_{1i})^2$ in the above equation – or in other words the squared components of $\mathbf{V}_1$ vectors). All of these squared $\mathbf{V}_1$ vector components are recorded in columns preceding the column in which total parameter identifiabilities are recorded. For a particular parameter, the pertinent element in each of these columns can be considered to be the square of the magnitude of the projection of a respective unit parameter vector onto the pertinent eigenvector. On the other hand, the pertinent element in the "total identifiabililty" column is the square of the magnitude of the projection of a respective unit parameter vector onto the totality of this space as spanned by all solution space eigenvectors.

Interesting and informative plots can be produced by importing this file into a spreadsheet or graphing program. For example the figure below shows part of a plot of total parameter identifiability, with the contribution from each solution space eigenvector demarcated by colour within each parameter-specific bar. It is apparent that some parameters (such as *par3*, *par8*, *par9* and *par11* are very identifiable, whereas others (such as *par13* to *par17*) are not.



Bar plot of identifiability of different parameters, coloured by contribution of different solution space eigenvectors.

The following should be noted:-

1. Weights used in computation of $\mathbf{Q}^{1/2}\mathbf{X}$ (or $\mathbf{X}^t\mathbf{Q}\mathbf{X}$) are those provided in the cited PEST control file.
2. If PEST is run in regularisation mode, observations and prior information equations belonging to regularisation groups are ignored.

3.28 SUPCALC

3.28.1 General

A common question, which is fundamental to using PEST's powerful SVD-assist inversion engine, is "how many super parameters should be employed in SVD-assisted calibration"? This question cannot be answered exactly. However a number of points can be made in consideration of this question. These include the following.

1. Moore and Doherty (2005) demonstrate that for any prediction made by a model, the error variance of that prediction first falls and then rises as the number of super parameters is increased. This corresponds to a decline in the null space contribution to predictive error variance and a simultaneous rise in the solution space contribution to predictive error variance with increasing number of super parameters. Predictive error variance is generally minimized at an intervening number of super parameters, this then defining the optimal number of super parameters to employ in calibrating a model prior to the making of that prediction. Fortunately, there is normally a range of singular values at which the error variance for most predictions is close to minimum, so finding the exact minimum is not critical. Also, the optimal number of super parameters is similar for most predictions, so that if a model is optimal for the making of one prediction, it is normally close to optimal for the making of most other predictions.
2. The upper limit on the number of super parameters that should be used in SVD-assisted calibration is set by that singular value whose ratio with the singular value of greatest magnitude is about 10^{-7}. (Singular values are normally arranged in order of increasing-to-decreasing value, and are always positive.) At this point numerical noise that accompanies finite-difference derivatives calculation is amplified to the point where parameter estimates are corrupted by that noise. The singular value number at which this occurs is normally far greater than that for which the error variance of any prediction is minimized.
3. Where Tikhonov regularisation is employed in SVD-assisted parameter estimation (with a target objective function that is set at an appropriately high level), more super parameters can be employed than would otherwise be the case, for the imposition of Tikhonov parameter constraints provides numerical stability and parameter reasonableness in its own way, compensating for possible loss of these benefits incurred by the use of too many super parameters.
4. It is often good to "err on the side of too many" super parameters rather than on the side of too few, this providing a means of insurance against the propensity for model nonlinearity to alter the boundary between the calibration solution and null spaces as parameter values are varied through the calibration process. As mentioned above, the deleterious effects of using too many super parameters can be circumvented by inclusion of Tikhonov regularisation in the inversion process. (Note that an easy way to add Tikhonov regularisation to an inverse problem is through the ADDREG1 utility.)

While conceptually appealing, computing the estimated error variance for a particular prediction prior to choosing the number of super parameters to employ in SVD-assisted parameter estimation is normally impractical. While an analysis of this kind can be undertaken using the PREDVAR1 or PREDVAR1A utility, predictive sensitivities (on which these utilities rely) are normally not available at the calibration stage of the model construction process (and may be expensive to compute anyway). However all is not lost, for a Jacobian matrix must have been computed prior to running SVDAPREP (this containing sensitivities to parameters of model outputs corresponding to members of the calibration dataset). If "predictions" can be formulated from sensitivities contained in this matrix, then PREDVAR-type analysis can be employed for providing assistance to the user in choosing the number of super parameters for use in the ensuing SVD-assisted calibration process. SUPCALC (which stands for "SUPer parameter CALCulator") was written to undertake such

analysis.

SUPCALC carries out singular value decomposition of either the $\mathbf{Q}^{1/2}\mathbf{X}$ or $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrices in order to obtain singular values and eigenvectors of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$. (Singular values of the latter matrix are the square of the former, while the eigenvectors of these two matrices are the same. Depending on the number of observations relative to the number of parameters employed in a particular inversion process, it may be more efficient to conduct SVD on one or the other of these matrices. SUPCALC's outcomes should be the same, irrespective of which matrix is decomposed.) Singular value decomposition yields a **V** matrix of orthogonal unit eigenvectors spanning parameter space, and a diagonal **S** matrix of corresponding singular values (arranged from highest to lowest), according to the formula:-

$$\mathbf{X}^t\mathbf{Q}\mathbf{X} = \mathbf{V}\mathbf{S}\mathbf{V}^t$$

Once it has carried out singular value decomposition of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$, SUPCALC then examines each of the eigenvectors corresponding to the columns of **V** in turn, starting with that corresponding to the highest singular value. It "pretends" that this is a prediction, and computes the fall in predictive error variance of this prediction (relative to its pre-calibration level) that is incurred through its inclusion as a super parameter. It also computes the rise in its error variance incurred through the fact that its estimation must take place on the basis of a noisy calibration dataset. If the fall in its error variance incurred through including it in the parameter estimation process (whereby it effectively moves from spanning part of the null space to spanning part of the solution space) is greater than the rise in its error variance incurred through the fact that its estimation is contaminated by measurement noise (the latter being amplified by the inverse of its singular value), then that super parameter is worth including in the parameter estimation process. However if the benefits of including it in the solution space are more than offset by losses due to error variance growth incurred by noise amplification (which becomes more and more likely as singular values fall), then the eigenvector (i.e. super parameter) is not worth including in the inversion process; nor will any super parameter corresponding to any higher singular value number, for the same reason. (Note, however, that there are some exceptions to this rule – see below).

As described elsewhere in this document, computation of predictive error variance requires that a covariance matrix of innate parameter variability be supplied (referred to as C(**p**) in this manual), together with a covariance matrix of measurement noise (referred to as C(**ε**)). When using SUPCALC, the former can be supplied through a parameter uncertainty file (see the PREDVAR and PREDERR suite of programs for documentation of this file type). Alternatively, SUPCALC can be told to compute the elements of C(**p**) itself on the basis of parameter bounds listed in the PEST control file which is the subject of its current analyses. SUPCALC computes the standard deviation of each parameter as one third of the distance between its lower and upper bound (or the log of this if a parameter is log-transformed). It also assumes that no correlation exists between parameters. (Note that when using this option for computation of C(**p**) elements, the user may need to take more care than he/she normally would in defining bounds in a PEST control file so that approximations employed in calculation of elements of C(**p**) in this manner do not result in C(**p**) elements which are too erroneous.)

For ease of use, SUPCALC assumes that the weighting strategy employed in a PEST control file results in a weight matrix **Q** which is proportional to the inverse of C(**ε**), the constant of proportionality being the "reference variance" σ^2_r. That is, it is assumed that:-

$$C(\boldsymbol{\varepsilon}) = \sigma^2_r\mathbf{Q}$$

where **Q** is defined through weights supplied in the PEST control file. Where a covariance matrix is provided for one or more observation groups in the PEST control file, the above relationship is still assumed to hold. (Do not worry too much if you consider this assumption to be violated in your PEST control file, for most "measurement noise" is in fact structural noise for which we can never know C(**ε**) anyway. Approximations such as this are an unavoidable part of modelling. Added to this is the fact that calculation of an optimal number of super parameters for use in SVD-assisted parameter estimation will only ever be approximate anyway. This matters little, especially when Tikhonov regularisation is also employed in the SVDA process as described above.) SUPCALC does not ask the user for the reference variance. Rather it asks him/her what measurement objective function he/she expects to achieve through the inversion process, and calculates the reference variance from that.

SUPCALC's estimate of the optimum number of super parameters to employ is not infallible. The user should therefore monitor the ensuing SVD-assisted inversion process carefully, looking for evidence that the number of super parameters may need to be altered. The following may serve as a guide when inspecting the progress of an SVD-assisted PEST run with this purpose in mind.

1. If matrix condition numbers are high (see the CND file), if the Marquardt lambda rises rather than falls as the parameter estimation process progresses, and if the measurement objective function "bounces around" rather than falls, then either the number of super parameters needs to be decreased or the Tikhonov target measurement objective function needs to be raised (or both).
2. If condition numbers are low (a few hundred or less) and you feel that PEST could have achieved a lower objective function than it did, the number of super parameters needs to be raised.

3.28.1 Using SUPCALC

SUPCALC is run by typing its name at the screen prompt. Unlike many PEST utilities it does not receive information through the command line. This is because the number of items of information that it requires would make this strategy somewhat cumbersome to implement. Hence it communicates with the user through a series of prompts, to which the user must respond with appropriate responses.

SUPCALC's first prompt is:-

```
Enter name of PEST control file:
```

Supply the name of an existing PEST control file, for which a complementary Jacobian matrix file already exists. the control file may or may not include Tikhonov regularisation. If it does, all regularisation observations and prior information equations are ignored in undertaking singular value decomposition of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ (or $\mathbf{Q}^{1/2}\mathbf{X}$), and in calculating the number of sustainable super parameters.

Next SUPCALC asks:-

```
Enter expected value of measurement objective function:
```

If you have not as yet calibrated the model, this will be a difficult question to answer. If you supply an answer which is too low, SUPCALC will probably inform you that you can employ more super parameters than is actually sustainable by the data. If your answer is too high then, based on SUPCALC's underestimation of the sustainable size of the calibration solution

space, it may suggest a number of super parameters that is somewhat too low. It is better to err on the side of optimism here and provide an objective function that may be too low. The use of Tikhonov regularisation in the SVDA process will provide numerical stability if this becomes necessary.

SUPCALC next asks:-

```
To conduct SVD on Q^(1/2)X  - enter 1
To conduct SVD on XtQX      - enter 2
Enter your choice:
```

SUPCALC output should not depend on your choice. If you have many observations, selection of the first option may lead to shorter SUPCALC run time as there is then no need for it to compute the $\mathbf{X}^t\mathbf{QX}$ matrix prior to undertaking singular value decomposition.

SUPCALC's next prompt is:-

```
Use uncertainty file or bounds to specify parameter variability?  [u/b]:
```

If you choose the "b" option, SUPCALC computes the standard deviation of every parameter as its range multiplied by 0.3. This factor is somewhat arbitrary. For a uniform probability distribution the standard deviation is actually the range divided by $\sqrt{12}$ (which is equivalent to multiplication by 0.288). For a normal distribution, if bounds signify a 95 percent confidence interval, the standard deviation is the range divided by 4 (which is equivalent to multiplication by 0.25). Furthermore, all parameters are assumed to be statistically independent (and thus C(**p**) – the covariance matrix of innate parameter variability – is assumed to be diagonal). The user should keep in mind the approximations that he/she implicitly makes through selection of the "b" option.

Alternatively, if the "u" option is chosen, the name of a parameter uncertainty file must be supplied in response to the following prompt:-

```
Enter name of parameter uncertainty file:
```

The format of a parameter uncertainty file is described elsewhere in this manual. Using this mechanism a C(**p**) matrix of arbitrary complexity can be supplied.

SUPCALC's final prompt is:-

```
Enter name for eigenvector gain/loss output file:
```

The format of this file will be discussed shortly.

Next, for each eigenvector represented in the **V** matrix (starting with that of highest corresponding singular value) SUPCALC computes the loss of uncertainty accrued through inclusion of this eigenvalue in the calibration solution space, and the gain in uncertainty incurred through the effect of measurement noise in estimating the parameter combination corresponding to this eigenvector once it is included in the solution space. It then sums the two of these. If the summation is negative (indicating a net loss of estimability of the parameter combination represented by the eigenvector), SUPCALC recommends that this eigenvector should not be included in the calibration solution space. The recommended minimum number of super parameters to include in the parameter estimation process is then declared to be one less than the number of that singular value.

Ideally, once a rise in total uncertainty is encountered in this fashion as singular value number is increased, the rise in total uncertainty will be even greater for eigenvectors corresponding to subsequent singular values. However where the C(**p**) matrix is non-diagonal, and where its diagonal elements are very different from each other, it is not impossible for losses in

uncertainty to follow gains. Hence SUPCALC supplies its minimum recommended number of super parameters as the singular value number corresponding to the last eigenvector for which inclusion in the solution space results in a net diminution of uncertainty.

As well as providing a value for the minimum recommended number of super parameters, SUPCALC also provides a recommendation for the maximum number of super parameters to employ. The maximum recommended number of super parameters corresponds to the last for which the singular value is above 10^{-7} of the first singular value. Both the minimum and maximum recommended number of super parameters are written to the screen by SUPCALC.

SUPCALC records the outcomes of calculations pertaining to all eigenvectors to its "eigenvalue gain/loss output file". An example of this output file follows.

```
Singular_value   fall_in_null_space_term    rise_in_soln_space_term     total_fall
    99.70827           1.183301              9.5014028E-03           1.173799
    35.44050           1.154450              2.6731237E-02           1.127718
    12.07416           1.084832              7.8462467E-02           1.006369
    8.386447           1.079208              0.1129642              0.9662440
    3.985802           1.035739              0.2376858              0.7980535
    1.738295           0.9812030              0.5449986              0.4362043
   0.7795562           0.9190415                1.215266             -0.2962249
   0.2791684           0.8709510                3.393538              -2.522587
   9.8457999E-02       0.8429289                9.622056              -8.779127
   3.0268715E-02       0.8483459                31.29860              -30.45025
```

A SUPCALC output file.

The first column of the SUPCALC output file lists singular values of $\mathbf{X}^t\mathbf{QX}$; these are listed in order of highest to lowest singular value. The fall in uncertainty of estimation of each eigenvector from its pre-calibration level (the latter uncertainty is determined by C(**p**) alone) is provided in the second column. The rise in uncertainty accrued from the fact that data on which basis this parameter combination is estimated is contaminated by measurement noise is provided in the third column. The final column contains the fall in uncertainty minus the rise in uncertainty; if this is negative, there is a net rise in uncertainty, from which it must be concluded that the eigenvector (equivalent to a super parameter) is not worth including in the parameter estimation process.

3.29 Super Observations

3.29.1 General

Where many observations are employed in the parameter estimation process (such as can regularly occur in the calibration of surface water models), the use of super observations in place of native observations can reduce PEST's memory requirements enormously, while inhibiting its performance to only a minor degree, if at all. Furthermore, in some circumstances, it can allow use of SVD-assisted parameter estimation to take place where this would otherwise be impossible because of the large amount of data that PEST must hold in memory as it computes super parameters from base parameters using a large Jacobian matrix calculated on the basis of the latter.

In addition to this, super observations can also provide a useful means of examining some important aspects of the calibration dataset, and of the information that resides within it.

3.29.2 Theory

Let the matrix **X** represent the action of a (linear) model on parameters **p**. Let the vector **h**

represent observations comprising the calibration dataset, and let the **ε** vector represent noise associated with this dataset. Then:

$$\mathbf{h} = \mathbf{Xp} + \boldsymbol{\varepsilon}$$

For simplicity, assume that noise is small enough to be ignored. Then:

$$\mathbf{h} = \mathbf{Xp}$$

Now let **X** be subjected to singular value decomposition such that:

$$\mathbf{X} = \mathbf{USV}^t$$

where (as is described elsewhere in this document) **U** is a matrix whose columns span the range space of **X**, **V** is a matrix whose columns span parameter space, and **S** is a diagonal matrix of singular values. Suppose that some of these singular values are zero, or near zero, and that these are partitioned into a submatrix $\mathbf{S}_2$ of **S**. (Note that $\mathbf{S}_2$ can be an empty matrix, for it is not fundamental to the use of super observations that the inverse problem of model calibration be ill-posed.) Thus:

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_2 \end{bmatrix}$$

X can thus be written as:

$$\mathbf{X} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{S}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^t \\ \mathbf{V}_2^t \end{bmatrix}$$

If $\mathbf{S}_2$ is ignored, this becomes:

$$\mathbf{X} = \mathbf{U}_1\mathbf{S}_1\mathbf{V}^t_1$$

so that:

$$\mathbf{h} = \mathbf{U}_1\mathbf{S}_1\mathbf{V}^t_1\mathbf{p}$$

If both sides of the above equation are multiplied by $\mathbf{S}^{-1}_1$, recalling that the columns of $\mathbf{U}_1$ are orthogonal, this becomes:-

$$\mathbf{S}^{-1}_1\mathbf{U}^t_1\mathbf{h} = \mathbf{V}^t_1\mathbf{p}$$

This equation is beautiful. Recalling that **S** is a diagonal matrix, it says that each linear combination of observations constituting the projection of the real-world dataset **h** onto a column of $\mathbf{U}_1$ is uniquely and solely informative of one, and only one, linear combination of parameters constituting a projection of real-world system properties **p** onto a column of $\mathbf{V}_1$. A "super parameter" (as estimated by PEST when it undertakes SVD-assisted parameter estimation) is actually the scalar projection of system properties **p** onto a single column of $\mathbf{V}_1$. A "super observation" (which constitutes a single element of the calibration dataset when these replace the original data elements encapsulated in **h** – which we shall do in a moment) is the scalar projection of the calibration dataset **h** onto a single column of $\mathbf{U}_1$. Thus each super observation is related to each corresponding super parameter by a scalar multiplier, this being the pertinent diagonal element of **S**.

Note that in calculating super observations, singular value decomposition is actually undertaken on the basis of the $\mathbf{Q}^{1/2}\mathbf{X}$ matrix rather than the **X** matrix, where **Q** is the matrix of observation weights pertaining to the current parameter estimation problem. Thus the weighting scheme associated with an observation dataset is accommodated when computing

super observations.

3.29.3 Some Uses of Super Observations

At the time of writing, two uses of super observations can be suggested. However more uses may emerge in the future.

The first use of super observations is in reducing the number of elements comprising the calibration dataset. For example in surface water model calibration, daily flow observations may be available at a number of gauging stations spread throughout a broad study area. Data from all of these gauging stations may be employed to underpin simultaneous estimation of parameters in watersheds upstream from all of them. Tikhonov regularisation may be used to constrain parameter values, and relationships between parameter values, to realistic values as assessed through expert knowledge of the study area. It is only through simultaneous, highly-parameterised, inversion of this kind that such relationships can be maintained over an entire study area at the same time as parameters are made to respect constraints on their values imposed by the necessity for the model to reproduce historical measurements of system state.

A calibration process, formulated in this way, may feature hundreds of parameters, and possibly tens of thousands of observations. Estimation of values for all of these parameters may be best effected using the SVD-assist inversion methodology. Experience in doing this has demonstrated that, even in data-rich environments, a relatively small number of parameter combinations (perhaps a few tens) can be estimated uniquely. Suppose that 50 super parameters are capable of unique estimation. The above equation tells us that, notwithstanding the size of the calibration dataset, only 50 linear combinations of observations house all information that can be extracted from the calibration dataset.

A suitable strategy for approaching the above parameter estimation problem may be to first reformulate it in terms of super observations rather than in terms of native observations. The number of super observations must be no greater than the number of native model parameters (and considerably smaller than the number of elements comprising the calibration dataset). The enormous reduction in the size of the calibration dataset that is achieved through the use of super observations can then make PEST's task much easier when undertaking SVD-assisted parameter estimation, for computation of super parameters then becomes a very rapid task indeed. Recall that when PEST undertakes SVD-assisted parameter estimation, super parameter computation must be undertaken at the start of the SVD-assisted parameter estimation process, and on each occasion that a base parameter hits its bounds.

A second use for super observations is in gaining insights into what elements of the native observation dataset are most informative of either individual model parameters, or of combinations of individual model parameters. For example, suppose that model calibration is taking place on the basis of a flow time series that is a few hundreds of elements long (say daily flows over a year or two). The calibration dataset may be comprised of the logs of these flows, or (what is almost equivalent), flows may be assigned weights that are inversely proportional to flow values. If $\mathbf{U}_1$ is calculated on the basis of the weighted Jacobian matrix arising from this calibration problem, the first few columns of $\mathbf{U}_1$ can be plotted as if they were flows. Those aspects of the flow time series that are most directly informative of different aspects of the system then become readily visible by inspecting the "orthogonally partitioned flow time series" corresponding to the first few columns of $\mathbf{U}_1$. The "different aspects of the system" referred to above constitute different linear combinations of parameters. These combinations can be obtained using the PCLC2MAT utility; each comprises a column of $\mathbf{V}_1$ corresponding to a column of $\mathbf{U}_1$.

3.29.4 The SUPOBSPREP Utility

A parameter estimation problem employing native observations can be transformed into a problem involving super observations using the SUPOBSPREP utility. Meanwhile the easier-to-use SUPOBSPAR utility builds files which list discrete super observations and the super parameters to which they are paired. The SUPOBSPREP utility is described below. The SUPOBSPAR utility is described later in the manual.

Use of SUPOBSPREP presupposes the existence of a PEST control file and corresponding JCO file. The following features of this pair of files are salient.

1. The JCO file need not correspond exactly to the PEST control file. If the latter includes prior information, this need not be included in the JCO file. However it must match the PEST control file in all other respects.
2. The command to run the model as cited in the PEST control file must end in "*.bat*", this being the name of a batch (PC) or script (UNIX) file.
3. The PEST control file can instruct PEST to run in either of parameter estimation, or regularisation modes.
4. If the PEST control file contains any prior information, this must belong to regularisation groups. However regularisation need not be restricted to prior information, for it is permissible for observations to belong to regularisation groups as well.

It is important to note that where PEST is run in regularisation mode, SUPOBSPREP builds super observations only from measurement observations, and not from any observations and/or prior information equations that comprise regularisation constraints (and hence belong to observation groups whose names begin with "regul"). The former are replaced by super observations in the new PEST control file written by SUPOBSPREP. The latter are transferred unchanged to the new PEST control file.

3.29.5 Running SUPOBSPREP

SUPOBSPREP is run by typing its name at the command line. It commences execution with the prompt:

```
Enter name of existing PEST control file:
```

As requested, provide the name of a PEST control file. SUPOBSPREP checks that a corresponding JCO file exists. As is discussed elsewhere in this manual, this is most easily produced by running PEST on the basis of the PEST control file with NOPTMAX set to -1 or -2 in that file. Note that it is good practice to check the integrity of a PEST input dataset using PESTCHEK before providing it to SUPOBSPREP.

Next SUPOBSPREP asks:

```
Enter number of super observations to build from this file:
```

Provide a number greater than zero. However if this number exceeds the number of non-regularisation observations present within the PEST input dataset, SUPOBSPREP will cease execution with an error message. *It will also cease execution with an error message if this number exceeds the number of adjustable parameters pertaining to the current parameter estimation problem.*

Now SUPOBSPREP gets a little technical. First it asks:

```
Enter clipping-enforced pre-compression weights range (<Enter> if 1E6):
```

The assignment of weights to super observations is a multi-step process. In the first of these steps the weight assigned to an observation is equated to the singular value with which it is associated. However singular value magnitudes can diminish rapidly with increasing super observation number. Suppose the default value of 1E6 is accepted in response to the above prompt. SUPOBSPREP will then ensure that the weight assigned to any super observation is no less than 1E6 of that assigned to the first super observation (which is always associated with the highest singular value) irrespective of the singular value associated with any super observation.

The next questions posed by SUPOBSPREP pertain to subsequent steps in super observation weights calculation.

```
Enable compression/expansion of super observation weights? (y/n):
```

Suppose that, after weights equal to respective singular values (with an appropriate low weight cutoff) have been assigned to super observations, the ratio of highest to lowest weight is 1E6. In many cases this range of weights will be too large, as this range may devalue the worth of all but the first few super observations. The ratio of highest-to-lowest super observation weight will be altered to a user-supplied value if the response to the above prompt is "y". This new ratio may, in fact, be larger or smaller than that which has already been calculated on the basis of singular values alone.

If the response to the above prompt is "y", SUPOBSPREP first asks for a new ratio of maximum to minimum weight:

```
Enter max/min super observation weight ratio:
```

and for the way in which weights compression (or expansion) should take place:

```
Undertake (n)atural or (l)ogarithmic compression to achieve this ratio:
```

Enter "n" or "l" as appropriate in response to the above prompt. In some circumstances, the latter approach may provide a more even spread of weights over large weight ranges than the former approach to weights compression/expansion.

SUPOBSPREP's final prompt in relation to weights computation is as follows. Note that the question below is posed whether or not weights compression/expansion takes place.

```
Enter minimum super-observation weight:
```

Suppose that a value of 1.0 is provided here. Then the minimum weight assigned to a super observation is 1.0, and all other weights are shifted upwards or downwards in proportion to this while maintaining ratios calculated through procedures discussed above.

SUPOBSPREP then asks for the name of the PEST control file that it must write. The prompt is:

```
Enter name for new super PEST control file:
```

Provide the name of a PEST control file as appropriate. (Note that if this filename does not possess an extension of "*.pst*" it will be rejected by SUPOBSPREP.)

Next SUPOBSPREP asks:

```
Enter name for super observation matrix file (<Enter> if none):
```

If a filename is provided in response to the above prompt, SUPOBSPREP will record the $\mathbf{U}_1$ matrix in PEST matrix file format. $\mathbf{U}_1$ has as many columns as there are super observations, and as many rows as there are non-regularisation observations in the original PEST dataset.

Individual columns can be extracted from this file using matrix utilities such as MATCOLEX.

Finally it asks:

```
Enter name for super parameter matrix file (<Enter> if none):
```

If a filename is provided in response to the above prompt, SUPOBSPREP will record the $\mathbf{V}_1$ matrix in PEST matrix file format. $\mathbf{V}_1$ has as many columns as there are super observations, and as many rows as there are non-regularisation observations in the original PEST dataset. The columns of $\mathbf{V}_1$ provide the parameter combinations which the columns of $\mathbf{U}_1$ respectively inform.

3.29.6 What SUPOBSCALC Does

SUPOBSCALC undertakes the following tasks. As it undertakes these tasks it informs the user, through its screen output, what it is doing.

1. It reads the PEST control file and associated Jacobian matrix file.
2. It forms the matrix $\mathbf{Q}^{1/2}\mathbf{X}$ where $\mathbf{Q}$ is the weight matrix associated with the current parameter estimation problem and $\mathbf{X}$ is the Jacobian matrix. (Note that native observations in the original PEST control file can be assigned individual weights, measurement covariance matrices, or a combination of these.)
3. It undertakes singular value decomposition of that component of $\mathbf{Q}^{1/2}\mathbf{X}$ that is associated with non-regularisation observations.
4. It writes a new PEST control file in which super observations replace native observations. Any observations and prior information equations assigned to regularisation groups in the original PEST control file are transferred directly to the new PEST control file.
5. It writes a Jacobian matrix pertaining to the new PEST control file. This Jacobian matrix file features derivatives of super observations with respect to model parameters (as well as derivatives of regularisation observations with respect to model parameters).
6. It writes a new model batch file (see below).
7. It writes an input file for the OBSCALC utility that is run as part of the modified model (see below).
8. Optionally it writes matrix files containing the $\mathbf{U}_1$ matrix and the $\mathbf{V}_1$ matrix (see above).

3.29.7 The New Model

As stated above, the new PEST dataset written by SUPOBSPREP features super observations instead of native observations (except where the latter belong to regularisation groups). Instruction files which instruct PEST how to read the model-generated equivalents of native observations from model output files are therefore absent from the new PEST control file. These numbers are actually read from model output files by a program named OBSCALC which is added to the model batch or script file by SUPOBSPREP.

OBSCALC (which is to super observations what the SVDA PARCALC utility is to super parameters), undertakes the following tasks.

1. It reads from model output files the model-generated equivalents to observations comprising the calibration dataset. It does this using the same instruction files as were provided in the original PEST input dataset.
2. It subtracts measured values from their model-generated counterparts.
3. It projects these differences into the columns of $\mathbf{U}_1$ as discussed above.
4. It records these projected differences on its output file. (These are super observation residuals.)

Because OBSCALC-calculated projected *residuals*, rather than projected observation *values*, are provided to PEST, the PEST control file written by SUPOBSPREP lists "observed" values of 0.0 for all super observations. Included in its "model input./output" section is an instruction file to read the OBSCALC output file. The latter is named *obscalc.out*; the associated instruction file is named *obscalc.ins*.

OBSCALC's input file (which contains the $\mathbf{U}_1$ matrix, as well as measurement values read from the original PEST control file) is named *obscalc.in*. This is written by SUPOBSPREP.

When run in order to build the new PEST dataset, SUPOBSPREP will cease execution with an error message if the model command (in the "model command line" section of the PEST control file) does not possess an extension of "*.bat*". SUPOBSPREP then assumes that this file is a batch file (in the PC environment) or a script file (in the UNIX environment). It modifies this file in the following ways.

1. It adds the command to run OBSCALC to the end of this file.
2. At the start of this file, it adds commands to delete model output files in which the model-generated equivalents to non-regularisation observations are recorded. Thus if for some reason the model fails to run, OBSCALC will not read old versions of these files, mistaking them for files just written by the model.

The new model batch or script file written by SUPOBSPREP is named *supobsbatch.bat*. This also constitutes the new model command recorded in the "model command line" section of the PEST control file written by SUPOBSPREP.

3.29.8 Some Features of the New PEST Dataset

Once a new PEST input dataset has been built by SUPOBSPREP, a calibration process based on super observations can be initiated by typing the command:

```
pest casename
```

at the command prompt, where ***casename*** is the filename base of the new PEST control file. Alternatively, the need to run the model many times for the purpose of finite-difference derivatives calculation during the first iteration of the parameter estimation process can be eliminated by running PEST using the command:-

```
pest casename /i
```

The "/i" switch is discussed elsewhere in this document. When started with this switch, PEST prompts for the name of a JCO file from which it reads derivatives for use on the first optimisation iteration of the parameter estimation process. In response to this prompt, supply the name of the JCO file written by SUPOBSPREP; this has the same filename base as that of the PEST control file, but possesses an extension of "*.jco*". Note that, no matter whether it is started with or without the "/i" switch, the SUPOBSPREP-computed JCO file is overwritten

by PEST during its first optimisation iteration.

If PEST is being run in regularisation mode, a new value will be required for the PHIMLIM variable, this being the target measurement objective function. When writing the new PEST control file, SUPOBSPREP transfers all control variables from the old one (including those in the "regularisation" section of this file, if such a section is included) to the new one. Obviously, with super observations employed instead of native observations, and with a singular-value-based weighting strategy for these super observations being employed, the target measurement objective function will need revision. Alternatively, set it to a suitably low value, possibly with a complementary FRACPHIM value of 0.1, and see what measurement objective function PEST can achieve when it runs on the basis of the super observation dataset. Having acquired this knowledge, a more appropriate target measurement objective function value can then be set for the next PEST run.

3.19.9 Using SVD-Assist with Super Observations

There is no reason why SVD-assisted parameter estimation cannot be conducted on a PEST input dataset which features super observations instead of native observations. This can be done simply through running SVDAPREP on the basis of the PEST input dataset written by SUPOBSPREP.

There is also nothing to stop a user from doing things the other way around. That is, SVDAPREP can be run on a native PEST input dataset to prepare a PEST input dataset for SVD-assisted parameter estimation. SUPOBSPREP can then be used to replace native observations appearing in this latter file by super observations. PEST can then be run on the basis of the PEST control file produced by SUPOBSPREP. If you do this, however, be sure to note the following.

1. When running SUPOBSPREP, remember to ask for no more super observations than there are super parameters defined in the SVD-assist PEST control file.
2. Before running SUPOBSPREP, set NOPTMAX to -1 or -2 in the SVDAPREP-generated super parameter PEST control file, and run PEST. PEST will then generate a super parameter Jacobian matrix and cease execution. If functionality for computation of super parameter derivatives on the basis of native parameter derivatives has been activated in the SVD-assisted PEST control file, PEST will undertake only one model run before writing the super parameter JCO file.
3. The super observation PEST control file produced by SUPOBSPREP will contain an identical "SVD-assist" section to that contained in the super parameter PEST control file written by SVDAPREP. This section will cite the original, pre-SVDAPREP, PEST control file (and associated JCO file) as the repository of native parameters (and their derivatives). Hence when PEST is run on the basis of this SUPOBSPREP-generated PEST control file, the sequence of BPA ("best base parameter value") files written by PEST that contain best native parameter values at all stages of the parameter estimation process, will possess a filename base which is the same as that of the original, pre-SVDAPREP, native parameter, PEST control file.

3.30 PESTCHEK

If PESTCHEK is run using the command:

```
pestchek case /s
```

where *case* is the filename base of a PEST control file, then PESTCHEK does not check any of the template and instruction files cited in the PEST control file for errors and for consistency with the PEST control file itself. In fact, it does not even check to see whether these files actually exist; instead it confines its checking to the PEST control file itself.

Nor does PESTCHEK issue any warning message, as these too are suppressed.

3.31 JCOCHEK

Program JCOCHEK reads a PEST control file and corresponding JCO file, checking that the two are compatible with each other. It is run using the command:

```
pestchek case
```

where *case* is the filename base of a PEST control file. JCOCHEK then reads *case.pst* as well as *case.jco*, checking that all adjustable parametes and observations cited in the first of these files are cited in the second, and that there are no observations or parameters cited in the JCO file that are not also cited in the PEST control file.

It is important to note however that special treatment is accorded to prior information. In particular:

1. If prior information is provided in the PEST control file, but is not cited in the JCO file, JCOCHEK declares the two files as being compatible, but issues a warning to this effect.
2. If prior information is cited in both files, JCOCHEK does not check that the sensitivities recorded in the JCO file are the same as the prior information parameter coefficients recorded in the PEST control file; it issues a warning to this effect.

3.32 SIMCASE

SIMCASE stands for "SIMplify CASE". It reads a PEST control file and a corresponding Jacobian matrix file. On the basis of information contained within these files it writes a new PEST control file and corresponding Jacobian matrix file. These files constitute a simplification of the original PEST input dataset, in that the following are omitted from it:

1. regularisation observations and prior information equations;
2. any parameters which are tied or fixed;
3. any parameter groups which contain no members;
4. any observation groups which contain no members;
5. any observation whose weight is zero.

In addition to this, PEST is instructed to run in "estimation" mode in the new PEST control file. Furthermore a dummy model command line is provided, as well as dummy template, instruction and model input./output filenames. (There is little use in citing the same template and instruction files if parameters and/or observations have been removed from the original PEST control file.)

While SIMCASE cannot be used as a basis for parameter estimation, it and its corresponding SIMCASE-generated JCO file can be employed by utility programs such as GENLINPRED, PREDVAR* and PREDUNC* for parameter/predictive error and uncertainty analysis. The fact that regularisation data has been removed from the PEST input dataset actually facilitates

use of these utility programs.

The integrity of a SIMCASE-produced PEST control file can be checked with the PESTCHEK utility, provided that PESTCHEK is run with the "/s" switch. Correspondence between the newly-created PEST control and Jacobian matrix files can be verified using the JCOCHEK utility.

3.33 JCOSUB

JCOSUB is run using the command:

```
jcosub jcofile1 jcofile2 jcofile3
```

where:

jcofile1 is the name of an existing Jacobian matrix file,

jcofile2 is the name of another existing Jacobian matrix file, and

jcofile3 is a new Jacobian matrix file written by JCOSUB.

Upon commencement of execution, JCOSUB reads both the first and the second of the existing JCO files cited on its command line. If any element of the second pertains to a parameter and observation named in the second, then the respective element of the first is overwritten by that of the second.

JCOSUB can be useful if you suspect that derivatives with respect to some parameters need improvement. You can create a new PEST control file in which all parameters but the worrisome ones are fixed. A new Jacobian matrix file (i.e. JCO file) can then be created by running PEST with NOPTMAX set to -2 on the basis of the second PEST control file. JCOSUB can then be run to make the necessary substitutions in the first JCO file. In order to save confusion, however, it is a good idea to remove all prior information from the second PEST control file. There is thus no danger of prior information sensitivities in the first JCO file from being overwritten.

3.34 New SVDAPREP Option

Previous versions of SVDAPREP prompted:

```
Write multiple BPA files?  [y/n] (<Enter> if "n"):
```

SVDAPREP has been upgraded to prompt:

```
Write multiple BPA, JCO, REI, none [b/j/r/n] files (<Enter> if "n")?
```

A request for multiple JCO and REI files will result in the JCOSAVEITN and REISAVEITN variables in the "control data" section of the PEST control file being set to "jcosaveitn" and "reisaveitn". Defaults are "nojcosaveitn" and "noreisaveitn" (or the absence of these variables). In requesting multiple BPA files the SVDA_MULBPA variable in the "svd assist" section of the PEST control file is set to 1.

If desired, the response to the above prompt can be comprised of multiple letters. For example a response of "jr" requests that both multiple JCO and REI files be saved. A response of "brj" (the order is unimportant) requests that all of the above file types be saved. Note, however, that a response of "n" must not be accompanied by any other letter. A response of "y" (equivalent to the response expected by the previous version of SVDAPREP when only multiple BPA files are requested) is interpreted as "b". It must also be supplied in conjunction with no other letter.

3.35 OBS2OBS

3.35.1 General

The OBS2OBS utility does for model outputs what the PAR2PAR utility does for model inputs. OBS2OBS reads user-specified numbers from files written by a model; it does this using instruction files (just as PEST does). New variables (named by the user) can be calculated from these model outputs on the basis of user-supplied equations. The values of these new variables (and, if desired, the values of the original model outputs) can be saved to a tabular output file. These can then be read by PEST and included within an observation dataset used for model calibration. PEST's reading of an OBS2OBS output file is made easier by the fact that OBS2OBS optionally generates an instruction set through which reading of this file can take place.

3.35.2 OBS2OBS Input File

Before running OBS2OBS an input file must be prepared. An example of an OBS2OBS input file is shown below.

```
* model output
model1.ins model1.out
model2.ins model2.out
* equations
hd1=head2-head1
cr1=log10(conc2/conc1)
* output
head1
head2
hd1
conc1
conc2
cr1
```

Example of an OBS2OBS input file.

An OBS2OBS input file must possess three sections, these being the "model output", "equations" and "output" sections. Each must begin with an appropriate header as shown above, this consisting of a "*" character, followed by a space, followed by the section name.

Blank lines can be inserted anywhere within an OBS2OBS input file; lines beginning with the "#" character are ignored, these being considered to be "commented out".

Sections within an OBS2OBS input file are now discussed in greater detail.

Model output

This section is similar to the "model input/output" section of a PEST control file. Each line within the "model output" section of an OBS2OBS input file must contain two entries. The second is the name of a model output file, while the first is the name of an instruction file which has been designed to read that model output file. As many such pairs of files can be supplied as desired (up to a large internally-set limit). Where a filename contains a space, it should be enclosed in quotes.

The same rules apply when supplying instruction files to OBS2OBS as apply when supplying them to PEST. These include the following.

1. An instruction filename cannot be repeated.
2. Observation names cited in instruction files must be unique.
3. Observation names must not contain spaces; their length is limited to 20 characters.

Equations

As the name implies, the "equations" section of an OBS2OBS input file must possess a set of equations. In each case the equation must be of the form:

new_variable=f(old_variables)

In the above symbolic notation the symbol *f*( ) denotes a function. This function can feature any observation read from a model output file (such observations being named in the instruction files that read them), or a *new_variable* whose value was assigned in a previous equation. However neither an *old_variable* nor a *new_variable* can be re-assigned using an equation. Like observations read from model output files, the name of each *new_variable* must be 20 characters or less in length, and cannot contain a space.

Examples of equations that can be used in definition of new variables are provided in documentation of PAR2PAR. Further examples are provided below.

```
ave_head = (head1+head2+head3)/3
penalty = max(head4-245.23,0,head5-324.5)
```

Operators and functions which can be used in equations featured in the "equations" section of an OBS2OBS input file are listed in the table below.

Operators	^, /, *, -,+, (,)
Functions	abs, acos, asin, atan, cos, cosh, exp, log, log10, sin, sinh, sqrt, tan, tanh, neg, pos, min, max, mod

Operators and functions that can be included in OBS2OBS equations.

The following should be noted.

1. In contrast to the PAR2PAR protocol, OBS2OBS will not allow equations to be continued onto the next line using a continuation character.
2. At the time of writing, an equation must not occupy more than 2000 characters of text.
3. Operators and variables can optionally be separated by spaces.

Output

The "output" section of an OBS2OBS input file must contain a list of variable names. These names can pertain to any variable read from a model output file (and hence named within an instruction file), or any variable calculated through equation evaluation. The names and values of these variables will be written to the OBS2OBS output file in the same order as that in which they are listed in the OBS2OBS input file. An example of an OBS2OBS output file is provided below.

```
head1                    3.456755261693154
head2                    4.324500931552616
hd1                      0.000000000000000
conc1                    30.45670000000000
conc2                    53.23432315324721
cd1                      22.77762050136947
```

Example of an OBS2OBS output file.

The following features of an OBS2OBS output file are salient.

1. The OBS2OBS output file contains two columns. No column headers are provided.
2. The first item of data on each line of an OBS2OBS output file is the name of a variable while the second item is the value of that variable.
3. The value of each variable is recorded using 16 significant figures, this being the maximum allowed by double precision machine number representation. Use of full precision in representation of numbers mitigates the potential for corruption of finite-difference derivatives.

3.35.3 Running OBS2OBS

OBS2OBS is run using the following command:

```
OBS2OBS infile outfile [insfile]
```

where:

infile is the name of an OBS2OBS input file (as described in the previous subsection);

outfile is the name of an output file to be written by OBS2OBS; and

insfile is the name of an instruction file to be (optionally) written by OBS2OBS.

The (optional) instruction file written by OBS2OBS reads the output file written by OBS2OBS, thus facilitating the addition of OBS2OBS output to a PEST calibration dataset. In this file observations are given the same names as variables read by OBS2OBS from model output files or named in an equation provided in the OBS2OBS input file. Adoption of this naming convention is not fundamentally necessary for use of OBS2OBS as part of a model run by PEST, as there is no need for PEST to use the same variable names as does OBS2OBS. The user is therefore free to alter an OBS2OBS-generated instruction file if he/she deems this to be necessary.

3.35.4 Some Uses of OBS2OBS

Observation post-processing

In many calibration contexts, model outputs and corresponding observations should be processed prior to being matched in formulation of separate components of an overall objective function. Steps should then be taken by the user to ensure that the contribution made by each component to the overall objective function neither dominates that of other objective function components, nor is dominated by them. Ideally processing should be undertaken in such a way as to "distil" from the observation dataset information pertaining to certain parameter combinations that would otherwise go unnoticed in the overall parameter estimation process. Examples include the differencing of head measurements in aquifers separated by an aquitard in order to better inform the vertical hydraulic conductivity of the material comprising the aquitard, and temporal differencing of measurements comprising a

time series in order to better inform storage parameters which govern temporal model output variations.

Making use of "soft data"

In many instances it is known that model outputs should lie within a certain range. It may not matter to the user where in the range these outputs fall, as long as the parameter estimation process is formulated in such a way as to discourage model outputs from straying outside of this range. Enforcement of constraints of this type requires use of a nonlinear penalty function - a function that enforces zero or minimal constraints if a model output is within certain limits, but that enforces rapidly growing constraints to the extent that these limits are violated.

OBS2OBS provides the user with a mechanism for achieving this aim. Suppose that it is desired that a model output *op1* be discouraged from exceeding *opmax* or undercutting *opmin*. Suppose also that the user does not feel that it would serve the parameter estimation process well to "observe" that *op1* is equal to the average of *opmin* and *opmax*. Two variables named *penalty_high* and *penalty_low* can then be defined within OBS2OBS as follows.

```
penalty_high = max(op1-opmax,0.0)
penalty_low = max(opmin-op1,0.0)
```

Each of these new variables will be zero unless the upper model output range is exceeded (in which case penalty_high will be non-zero, or unless the lower model output range is transgressed (in which case *penalty_low* will be non-zero). These two variables can then be matched to "observed" values of zero in the PEST control file. The weight given to these observations should then be such as to provide a strong disincentive for these bounds to be transgressed.

Further sophistication can be added by raising *penalty_high* and *penalty_low* to an appropriate power (greater than unity) to strengthen the onset of penalty enforcement. Alternatively, if a discontinuous penalty function is not warranted, a continuous penalty function defined as follows may be better.

```
mean = (opmax+opmin)*0.5
meandiff = opmax-mean
penalty = ((op1-mean)/meandiff)^2
```

Penalty will be zero when *op1* is equal to the mean of *opmax* and *opmin*. It will be equal to one when *op1* is equal to *opmax* or *opmin*. It will rise rapidly if further bounds transgression takes place. (In some circumstances logarithmic transformation of model outputs and bounds may be warranted in implementing this procedure).

Parameter bounds enforcement

PEST allows for "hard" bounds to be emplaced on parameters; these bounds will not be transgressed during the parameter estimation process (unless they are applied to a child parameter that is tied to a fixed parameter, with only the latter being estimated). Such rigid "on" or "off" bounds enforcement may not be appropriate in some modelling contexts. Furthermore a practical problem is encountered when undertaking SVD-assisted parameter estimation in that if a base parameter encounters its bound, it is stuck there for the remainder of the parameter estimation process.

An appropriate response to these situations may be to employ soft bounds in addition to hard bounds. Prior information (in which a parameter is assigned a preferred value with a penalty incurred through deviation from this preferred value) could be used as a mechanism for imposition of soft bounds. However this may provide information to the parameter estimation process which, in the modeller's opinion, may misguide that process. Another approach may be to use parameter thresholds in the same manner as that shown above for observations. The upper and lower parameter thresholds (both of which should be within the bounds associated with that parameter) defined parameter values at which a linear (or nonlinear) penalty begins to be enforced in the manner shown above. Of course if PEST is still determined to send the parameter to its upper or lower bound notwithstanding the increasing disincentive to do so as that bound is approach (if the threshold and penalty power function are properly set), than bounds enforcement takes place in the usual way.

If implementing soft bounds enforcement, PEST should be asked to write an additional "model input file" on the basis of a new, user-supplied template file. This model input file should simply list current values for parameters for which soft bounds enforcement is to be applied. This file can then be read as a "model output file" by OBS2OBS using an appropriate instruction file.

3.35.5 Some Notes on Using OBS2OBS

OBS2OBS is normally used to supplement an existing PEST input dataset. Hence the model output file read by OBS2OBS may also be read by PEST. The OBS2OBS output file is then added to the list of model output files recorded in the "model input/output" section of the PEST control file (together with the instruction file which is used to read this OBS2OBS output file – possibly generated by OBS2OBS itself in the manner described above). Alternatively, the user can eliminate the need for PEST to read "raw" and OBS2OBS-processed observations from two different files. This is made possible by the fact that OBS2OBS can record to its output file the values of model outputs that it reads from model output files. Thus these, together with OBS2OBS-processed outcomes of these model outputs, can be read from the one file, this being the OBS2OBS output file. Model outputs thus "pass through" OBS2OBS, undergoing no change in the process.

If OBS2OBS is used as a model postprocessor then the command to run OBS2OBS must be included in the model batch file (after the command to run the model). As stated elsewhere in PEST documentation, it is a good idea to delete the input file of any executable program that runs in the model batch file (using an appropriate deletion command early in the batch file) if that input file is written by another executable program cited in the batch file. Hence if this input file is not produced because of execution failure in the program that is supposed to write it, the program which is supposed to read it will crash. PEST will then detect a model failure condition instead of reading the outcomes of model component calculations that have taken place on the basis of files that are left over from a previous model run. This advice is just as salient when using OBS2OBS, except under some circumstances. These include the following.

1. If the model output file that is read by OBS2OBS is also read by PEST, then there is no need to delete this file using a batch file command, as PEST will delete this file itself prior to running the model.
2. The OBS2OBS input file must not be deleted if it contains parameter values directly written by PEST using a template file (this being done to enforce soft parameter bounds in the manner described above). Deletion of this file early in the model batch

process will ensure its absence when OBS2OBS needs to read it. However if the user feels uncomfortable in straying from the established principle of deleting all input files used by model post-processors (this being a worthy principle), then he/she should include in the model batch file a command to copy the PEST-generated parameter list file to another file, and instruct OBS2OBS to read the latter file. The command to delete this file can then be included towards the top of the batch file.

3.36 SUBREG1

As the name suggests, SUBREG1 performs the opposite task to that of ADDREG1, in that it subtracts regularisation from a PEST control file. It is run using the command:

```
SUBREG1 infile outfile
```

where:

infile is the name of an existing PEST control file; and

outfile is the name of a new PEST control file.

Note that the "*.pst*" extension can be omitted from either or both of the input or output PEST control files if desired.

SUBREG1 undertakes the following tasks.

1. It alters the PESTMODE variable to "estimation".
2. It removes from the "observation groups" section of the PEST control file all observation groups whose names begin with "regul"; the value of the NOBSGP variable in the "control data" section is adjusted accordingly.
3. It removes from the "observation data" section of the PEST control file, all observations that belong to observation groups whose names begin with "regul"; the value of the NOBS variable in the "control data" section is adjusted accordingly.
4. It removes from the "prior information" section of the PEST control file all prior information equations that belong to observation groups whose names begin with "regul"; the value of the NPRIOR variable in the "control data" section is adjusted accordingly.

SUBREG1 does not remove or alter any instruction files. Hence if observations are removed from the "observation data" section of the PEST control file, an incompatibility will exist between the new PEST control file and the names of instruction files cited in that file. The task of rectifying this incompatibility belongs to the user. Do not run PEST until PESTCHEK gives you the all clear.

3.37 SUPOBSPAR

Super parameters and super observations are defined and discussed in the documentation to the SUPOBSPREP utility. SUPOBSPAR makes calculation of these relatively simple. In restricting its activities to computation only of these complimentary combinations of observations on the one hand, and the parameters which they inform on the other hand, SUPOBSPAR is easier to use than SUPOBSPREP and less restrictive in the demands it makes on the existing PEST input dataset. In particular, the number of parameters cited in a PEST control file which it reads can exceed the number of observations cited in that file; or the opposite may occur.

SUPOBSPAR is run using the command:

```
SUPOBSPAR casename N obsmatfile parmatfile
```

where:

casename is the filename base of an existing PEST control file;

N is the number of super observations and corresponding super parameters to compute;

obsmatfile is the name of a matrix file to which SUPOBSPAR will write super observations; and

parmatfile is the name of a matrix file to which SUPOBSPAR will write super parameters.

A Jacobian matrix file (i.e. a JCO file) must accompany the PEST control file. If this is not the case, SUPOBSPAR will cease execution with an appropriate error message. The PEST control file may or may not instruct PEST to run in regularisation mode. If it does, SUPOBSPAR will ignore regularisation prior information, for regularisation devices such as these should not be included in the computation of super parameters and super observations as the latter should reflect the information content of the calibration dataset alone.

Observation and parameter vectors defining super observations and super parameters are recorded in PEST matrix file format. Individual columns can be extracted from such files using the MATCOLEX utility. However if less than eight super observations/parameters are requested, then wrapping of rows within these files will not occur and column extraction may not be necessary; in most cases a user will only be interested in the first few super observations and super parameters anyway. The format employed by PEST matrix files is such that the elements of the matrix are recorded first (with row-wrapping as necessary), followed by lists of row names and column names. Row names correspond to observation names where super observations are recorded and to parameter names where super parameters are recorded. These names can be cut and pasted alongside column elements for easy linkage of element values to element names. Also, where parameters have a spatial or temporal connotation, elements of super observations and/or super parameters can be plotted against space and time so that their patterns can define the spatial/temporal distribution of observation information content on the one hand, and the spatial/temporal distribution of parameter recipients of that information on the other hand. Altnertively, bar charts of the observation/parameter content of super observations and super parameters can be plotted as histograms in software such as Microsoft EXCEL.

3.38 SUPOBSPAR1

3.38.1 General

SUPOBSPAR1 is a variant of SUPOBSPAR in that it calculates vectors comprising super parameters and super observations through undertaking singular value decomposition of the weighted Jacobian matrix. However it automatically applies a Karhunen Loeve (KL) transformation to parameters prior to undertaking singular value decomposition. The model matrix on which singular value decomposition is performed is altered as a consequence, as it effectively operates on KL-transformed parameters rather than native model parameters. Once this has been done, back-transformation of KL-transformed super parameters is undertaken so that they pertain to native model parameters rather than KL-transformed parameters.

Theoretically, KL-transformation of parameters prior to their estimation constitutes the best way to ensure optimality of inversion undertaken though singular value decomposition as it leads to minimum error variance parameter estimates, and minimum error variance predictions made by the calibrated model. Inversion based on KL-transformed parameters takes account of the information content of expert knowledge as encapsulated in the prior parameter covariance matrix. Super observations and super parameters determined in this way are therefore the "natural" super parameters and super observations that emerge from an inversion process where information that is available in expert knowledge on the one hand and measurements of system state on the other hand are optimally blended.

3.38.2 Theory

Let C(**k**) represent the covariance matrix associated with the prior probability distribution of a parameter set **k**. Through singular value decomposition, the **F** and **E** matrices (the latter being diagonal) can be defined through the equation:

$$C(\mathbf{k}) = \mathbf{F}\mathbf{E}\mathbf{F}^t$$

The KL-transformed parameter set **j** is defined through:

$$\mathbf{j} = \mathbf{E}^{-1/2}\mathbf{F}^t\mathbf{k}$$

so that:

$$\mathbf{k} = \mathbf{F}\mathbf{E}^{1/2}\mathbf{j}$$

Applying the classical formula for propagation of covariance readily shows that:

$$C(\mathbf{j}) = \mathbf{I}$$

Let the action of a model be represented by the matrix **Z**. Let observations comprising the calibration dataset be represented by **h**, and let **ε** represent measurement noise. Then:

$$\mathbf{h} = \mathbf{Z}\mathbf{k} + \boldsymbol{\varepsilon}$$

From the above it follows that:

$$\mathbf{h} = \mathbf{Z}\mathbf{F}\mathbf{E}^{1/2}\mathbf{j} + \boldsymbol{\varepsilon}$$

Let us select a weight matrix **Q** such that:

$$\mathbf{Q} = C^{-1}(\boldsymbol{\varepsilon})$$

Selection of such a weight matrix promulgates optimal inversion in that it minimizes the contribution that measurement noise makes to the error variance of estimated parameters and predictions which depend on them. Premultiplying the model equation by $\mathbf{Q}^{1/2}$ we obtain:

$$\mathbf{Q}^{1/2}\mathbf{h} = \mathbf{Q}^{1/2}\mathbf{Z}\mathbf{F}\mathbf{E}^{1/2}\mathbf{j} + \boldsymbol{\eta}$$

where:

$$\boldsymbol{\eta} = \mathbf{Q}^{1/2}\boldsymbol{\varepsilon}$$

Obviously:

$$C(\boldsymbol{\eta}) = \mathbf{I}$$

In most circumstances of practical interest **Q** is a diagonal matrix, with elements equal to weights supplied in the PEST control file. Now let us perform singular value decomposition on the modified model matrix such that:

$$\mathbf{Q}^{1/2}\mathbf{Z}\mathbf{F}\mathbf{E}^{1/2} = \mathbf{U}\mathbf{S}\mathbf{V}^t$$

If measurement noise is ignored, it follows from the model equation that:

$$\mathbf{Q}^{-1/2}\mathbf{h} = \mathbf{USV}^t\mathbf{E}^{-1/2}\mathbf{F}^t\mathbf{k} = \mathbf{USW}^t\mathbf{k}$$

"Super observations" computed by SUPOBSPAR1 are the columns of **U** while super parameters computed by SUPOBSPAR1 are the rows of $\mathbf{W}^t$. The former directly and completely inform the latter.

3.38.3 Running SUPOBSPAR1

SUPOBSPAR1 is run using the command:

```
SUPOBSPAR1 casename uncertfile N obsmatfile parmatfile
```

where:

casename	is the filename base of an existing PEST control file;
uncertfile	is the name of a parameter uncertainty file – this contains the C(**k**) matrix;
N	is the number of super observations and corresponding super parameters to compute;
obsmatfile	is the name of a matrix file to which SUPOBSPAR1 will write super observations; and
parmatfile	is the name of a matrix file to which SUPOBSPAR1 will write super parameters.

3.39 INFSTAT1

3.39.1 General

INFSTAT1 performs a similar function to INFSTAT. The difference is in how it handles regularisation.

Where an inverse problem is ill-posed, INFSTAT cannot invert the sensitivity matrix required for calculation of its information statistics. This situation can be remedied by including regularisation prior information in the PEST control file. Presumably PEST has calculated appropriate weights for this prior information during a previous inversion exercise. (INFSTAT reads files created by PEST during this exercise to obtain these weights).

Instead of using Tikhonov regularisation as the means of creating parameter uniqueness, INFSTAT1 employs subspace methods as encapsulated in singular value decomposition (SVD). In fact, if any prior information is contained within a PEST control file, INFSTAT1 ignores it. The user informs INFSTAT1 of the number of singular values at which SVD-truncation must take place. In doing this he/she sets the dimensionality of the inversion solution space. INFSTAT1 then reformulates sensitivity matrices so that parameter projections onto this space are estimated. If the dimensionality of the solution space is too large (INFSTAT1 decrees it to be too large if the ratio of the smallest to largest post-truncation eigenvalue is less than 5E-7), INFSTAT1 informs the user of this, and then ceases execution. Thus all matrices required for calculation of influence statistics are guaranteed to be invertible.

The SUPCALC utility can be used to obtain an optimal solution space dimensionality. Use

the lower of the two values suggested by SUPCALC.

As stated above, when SVD is employed as a regularisation device, the inverse problem is re-formulated to infer the projection of a real-world parameter set onto the solution space, rather than this parameter set itself. The direction cosine between an individual parameter and its projection into the solution space is referred to as the "identifiability" of that parameter herein; this can be calculated using the IDENTPAR utility (and GENLINPRED). For many parameters the identifiability may not be very high. Where this is the case, the estimated value of the parameter may have a large error due to the high projection of that parameter onto the null space. INFSTAT1 does not concern itself with this. Each parameter cited in its output file must be interpreted as the projection of the pertinent parameter into the solution space.

3.39.2 Running INFSTAT1

INFSTAT1 is run using the command:

```
infstat1 casename outfile nsing
```

where:

casename is the filename base of a PEST control file;

outfile is the INFSTAT1 output file, and

nsing is the singular value truncation limit.

If NSING is supplied as greater than the number of adjustable parameters or the number of non-regularisation observations cited in a PEST control file, it is automatically reduced to the lower of these two numbers. The eigenvalue ratio at the user-supplied truncation threshold is written to both the screen and to the INFSTAT1 output file.

It is important that PEST has been run prior to using INFSTAT1 as a Jacobian matrix corresponding to the nominated PEST control file must be present. If PEST is run specifically to obtain this matrix, then NOPTMAX must be set to -1 and not -2 in that file so that a residuals file has also been recorded. INFSTAT1 uses these residuals in calculation of its influence statistics.

See also the SSSPTAT utility. Like INFSTAT and INFSTAT1, this utility establishes the information content of observations used in the inversion process. However this utility takes into account the reduced identifiability of some parameters due to non-zero null space projection of these parameters.

3.40 SSSTAT

3.40.1 Introduction

SSSTAT stands for "subspace statistics". It can be used to study the outcomes of a parameter estimation process, highly parameterized or otherwise. Statistics pertaining to the information content of individual members of an observation dataset are calculated, as well as statistics pertaining to the estimability and post-calibration error variance of parameters. The user is also given access to the normalized sensitivity matrix through which model outputs corresponding to observations are calculated from parameters, and to the matrix though which estimated parameters are calculated from observations.

3.40.2 Theory and Concepts

Let the vector **h** represent observations comprising the calibration dataset and let the vector **ε** denote the errors associated with those measurements. In the calibration context this will include both measurement error and model structural error. Let **k** denote the parameter set employed by the model. Let the matrix **X** represent the action of the (linearized) model. Then

$$\mathbf{h} = \mathbf{X}\mathbf{k} + \boldsymbol{\varepsilon} \qquad (3.40.1)$$

We now normalize both model outputs and model parameters. The former are normalized with respect to the measurement noise that is associated with them, while the latter are normalized with respect to their innate variability. Define the weight matrix **Q** such that:

$$\mathbf{Q} = \mathbf{C}^{-1}(\boldsymbol{\varepsilon}) \qquad (3.40.2)$$

Let the vector j represent a transformed parameter set, calculated from **k** through the equation:

$$\mathbf{j} = \mathbf{E}^{-1/2}\mathbf{F}^{t}\mathbf{k} \qquad (3.40.3)$$

so that:

$$\mathbf{k} = \mathbf{F}\mathbf{E}^{1/2}\mathbf{j} \qquad (3.40.4)$$

F and **E** are defined through singular value decomposition of the prior parameter covariance matrix C(**k**) (which also can be considered as the covariance matrix of innate parameter variability) through:

$$C(\mathbf{k}) = \mathbf{F}\mathbf{E}\mathbf{F}^{t} \qquad (3.40.5)$$

Obviously:

$$C(\mathbf{j}) = \mathbf{I} \qquad (3.40.6)$$

Hence the transformed parameters **j** are normalized with respect to their innate variability. That is, they have been subjected to so-called Kahunen-Louve (KL) transformation. Where normalization of model outputs and model parameters is undertaken, the model equation can be written:

$$\mathbf{f} = \mathbf{Z}\mathbf{j} + \boldsymbol{\tau} \qquad (3.40.7)$$

where:

$$\mathbf{Z} = \mathbf{Q}^{1/2}\mathbf{X}\mathbf{F}\mathbf{E}^{1/2} \qquad (3.40.8)$$

$$\mathbf{f} = \mathbf{Q}^{1/2}\mathbf{h} \qquad (3.40.9)$$

and:

$$\boldsymbol{\tau} = \mathbf{Q}^{1/2}\boldsymbol{\varepsilon} \qquad (3.40.10)$$

Obviously:

$$C(\boldsymbol{\tau}) = \mathbf{I} \qquad (3.40.11)$$

SSSTAT bases its calculation of inversion statistics on the normalized model equation, i.e. equation (3.40.7).

SSSTAT works at two levels, the hiatus between these levels being marked by whether or not singular value decomposition (SVD) of the **Z** matrix has been undertaken. Singular value decomposition of **Z** leads to calculation of matrices **U**, **S** and $\mathbf{V}^{t}$ defined through the usual SVD equation:

$$\mathbf{Z} = \mathbf{USV}^t \qquad (3.40.12)$$

Prior to undertaking SVD of the **Z** matrix, SSSTAT calculates the diagonal elements of the matrix $\mathbf{ZZ}^t$ and the diagonal elements of the matrix $\mathbf{Z}^t\mathbf{Z}$. From equation (3.40.12) it follows that:

$$\mathbf{ZZ}^t = \mathbf{US}^2\mathbf{U}^t \qquad (3.40.13)$$

and that:

$$\mathbf{Z}^t\mathbf{Z} = \mathbf{VS}^2\mathbf{V}^t \qquad (3.40.14)$$

Note also that if measurement noise is ignored, then, from (3.40.7):

$$C(\mathbf{f}) = \mathbf{Z}C(\mathbf{j})\mathbf{Z}^t = \mathbf{ZZ}^t \qquad (3.40.15)$$

Hence $\mathbf{ZZ}^t$ is the covariance matrix that denotes the variability of model outputs **f** as this arises from the natural variability of **j**. This can be compared with the variability of **j** induced by measurement/structural noise, of which the covariance matrix is **I**. The diagonal elements of $\mathbf{ZZ}^t$ thus mark the variability of individual model outputs (that correspond to observations) arising out of natural parameter variability. As such, it is a measure of their information content with respect to parameters, for high parameter-induced variability indicates high information content with respect to the parameters that induce that variability.

$\mathbf{Z}^t\mathbf{Z}$ can be viewed in a number of ways. The diagonal elements of this matrix are the composite sensitivities of parameters, scaled by their innate variability. This is thus closely related to the CSS statistic of Hill and Tiedeman (2007), but is a little more theoretically based. If the inverse problem is well posed, $(\mathbf{Z}^t\mathbf{Z})^{-1}$ is the post-calibration parameter covariance matrix. However if it is not well posed this matrix cannot be formed. Because the diagonal elements of $(\mathbf{Z}^t\mathbf{Z})^{-1}$ denote individual post-calibration parameter uncertainty, the diagonal elements of $\mathbf{Z}^t\mathbf{Z}$ can be seen as loosely denoting post-calibration "parameter certainty". Low values of post-calibration "parameter certainty" denote low ability to be inferred through the calibration process. The post-calibration uncertainty of such parameters must therefore be constrained by expert knowledge rather than information that is resident in the calibration dataset.

The diagonal elements of $\mathbf{Z}^t\mathbf{Z}$ can be viewed from another perspective. First consider the matrix $\mathbf{V}_1\mathbf{V}^t_1$, where the "1" subscript indicates a partitioning of **V** based on non-zero (or non-near-zero) singular values. The vectors comprising the columns of $\mathbf{V}_1$ thus span the inversion solution space. Those comprising the orthogonal complement of $\mathbf{V}_1$, i.e. $\mathbf{V}_2$, span the inversion null space. Thus:

$$\mathbf{V} = [\mathbf{V}_1 \ \ \mathbf{V}_2] \qquad (3.40.16)$$

The matrix $\mathbf{V}_1\mathbf{V}^t_1$ denotes an orthogonal projection of (unknown) real model parameters onto the inversion solution space. As such it comprises the so-called "resolution matrix" that emerges from inversion based on truncated singular value decomposition. The diagonal elements of $\mathbf{V}_1\mathbf{V}^t_1$ are thus the diagonal elements of the resolution matrix. They are also the "parameter identifiabilities" of Doherty and Hunt (2009). These range between 0 and 1. Parameters with an identifiability of zero are completely inestimable through the inversion process because they lie in the inversion null space. Parameters with an identifiability of 1 are completely estimable as they lie entirely in the inversion solution space. The identifiability of a parameter is in fact the cosine of the angle between it and its projection onto the solution space.

Based on these considerations, as well as the characterization of $\mathbf{Z}^t\mathbf{Z}$ provided through

equation (3.40.14), the matrix $\mathbf{Z}^t\mathbf{Z}$ can be viewed as the sum of weighted direction cosines between the real-world (and unknown) scaled parameter vector **j** and its projection onto each of the vectors $\mathbf{v}_i$ comprising the columns of $\mathbf{V}_1$ which span the inversion solution space. The weight in each case is the square of s_i, this being the singular value associated with $\mathbf{v}_i$. The higher is this singular value, the less is the estimation of that parameter contaminated by measurement noise. See below.

Where inversion is undertaken through SVD, the calibrated scaled parameter set $\underline{\mathbf{j}}$ is calculated through the equation:

$$\underline{\mathbf{j}} = \mathbf{V}_1\mathbf{S}_1\mathbf{U}^t\mathbf{f} = \mathbf{G}\mathbf{f} \qquad (3.40.17)$$

Moore and Doherty (2005) show that the post-calibration error variance of any parameter is the sum of two terms. The first is the null space contribution to error and the second is the solution space contribution to error. It is easily shown from (3.40.17) and (3.40.7) that:

$$\underline{\mathbf{j}} = \mathbf{V}_1\mathbf{V}^t{}_1\mathbf{j} + \mathbf{V}^t{}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t\boldsymbol{\tau} \qquad (3.40.18)$$

so that:

$$\underline{\mathbf{j}} - \mathbf{j} = -\mathbf{V}_2\mathbf{V}^t{}_2\mathbf{j} + \mathbf{V}^t{}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t\boldsymbol{\tau} \qquad (3.40.19)$$

Recalling that C(**j**) and C($\boldsymbol{\tau}$) are both the identity matrices, it follows that:

$$C(\underline{\mathbf{j}} - \mathbf{j}) = \mathbf{V}_2\mathbf{V}^t{}_2 + \mathbf{V}_1\mathbf{S}^{-2}{}_1\mathbf{V}^t{}_1 \qquad (3.40.20)$$

After computing $\mathbf{Z}\mathbf{Z}^t$ and $\mathbf{Z}^t\mathbf{Z}$, SSSTAT undertakes singular value decomposition of **Z**. Using a method identical to that employed by the SUPCALC utility, it then evaluates the optimal SVD truncation point; that is, it computes the optimal dimensionality of the solution space. It does this by looking at the pre- and post-calibration error variances of each of the coefficients of the vectors $\mathbf{v}_i$ comprising the columns of **V** estimated through the inversion process. The pre-calibration error variance of each of these coefficients arises from expert knowledge alone. This is easily shown to be 1.0 for each coefficient. The post calibration error variance of each coefficient is equal to the inverse of the square of the respective singular value. The solution space is deemed to end and the null space is deemed to begin where post-calibration error variance exceeds pre-calibration error variance. This occurs where singular values fall below 1.0.

After determining the optimal truncation point, SSSTAT lists on its output file all of the singular values arising from SVD of the **Z** matrix. It then lists the diagonal elements of $\mathbf{V}^t{}_1\mathbf{V}_1$ and $\mathbf{V}^t{}_2\mathbf{V}_2$. As stated above, the former are the identifiabilities of the scaled parameters comprising the vector **j**. The latter complement these; these are the cosines of the angles between scaled parameters and their projections onto the null space; $\mathbf{V}^t{}_1\mathbf{V}_1$ and $\mathbf{V}^t{}_2\mathbf{V}_2$ sum to **I**. The diagonal elements of $\mathbf{V}^t{}_2\mathbf{V}_2$ can also be considered as the post-calibration null space components of error variance associated with each parameter. See Moore and Doherty (2005).

SSSTAT next computes and lists the diagonal elements of $\mathbf{V}^t{}_1\mathbf{S}^2{}_1\mathbf{V}_1$ and $\mathbf{V}^t{}_1\mathbf{S}^{-2}{}_1\mathbf{V}_1$. As discussed above, if the solution space comprises the whole of parameter space, the former quantities are equal to the diagonal elements of $\mathbf{Z}^t\mathbf{Z}$. If not, they will be equal to or less than these. If the solution space comprises the entirety of parameter space, $\mathbf{V}^t{}_1\mathbf{S}^{-2}{}_1\mathbf{V}_1$ will be equal to $(\mathbf{Z}^t\mathbf{Z})^{-1}$. If not, it will be equal to the solution space component of the covariance matrix of post-calibration scaled parameter error. Each diagonal elements of this matrix is thus the post-calibration null space contribution to the error variance of the scaled parameter comprising the respective element of the vector **j**. For each scaled parameter the sum of this

diagonal element and the respective diagonal element of $\mathbf{V}^t_1\mathbf{S}^{-2}_1\mathbf{V}_1$ is the total post-calibration error variance. SSSTAT lists these for all scaled parameters.

Finally, SSSTAT computes the diagonal elements of the matrix $\mathbf{U}^t_1\mathbf{S}_2\mathbf{U}_1$. Where **U** is not truncated through inversion (which can occur if, for example, the number of parameters is greater than the number of observations, or if the inversion problem is well posed), the $\mathbf{U}^t_1\mathbf{S}_2\mathbf{U}_1$ matrix is equal to the $\mathbf{ZZ}^t$ matrix. If not, the diagonal elements of $\mathbf{U}^t_1\mathbf{S}_2\mathbf{U}_1$ will be slightly less than those of $\mathbf{ZZ}^t$. These provide a measure of variability of weight-scaled model outputs arising from variability of solution space components of scaled parameters. As such the diagonal elements of $\mathbf{U}^t_1\mathbf{S}_2\mathbf{U}_1$ are the error variances of respective model outputs arising from solution space scaled parameter variabilities.

3.40.3 Using SSSTAT

SSSTAT is run by typing its name at the screen prompt. Typical prompts and responses are as follows.

```
Enter name of PEST control file: pf12.pst
Enter expected value of measurement objective function: 10030

Use uncertainty file or bounds to specify parameter variability?  [u/b]: b

Enter name for SSSTAT output file: temp.dat
Enter name for G matrix output file: tempg.mat
Enter name for Z matrix output file: tempz.mat

- reading PEST control file pf12.pst....
- file pf12.pst read ok.

- reading Jacobian matrix file pf12.jco....
- file pf12.jco read ok.

- transforming Jacobian matrix....
- carrying out singular value decomposition of Q^(1/2)XFE(1/2)....
- optimal truncation point = 4 singular values.
- forming G matrix....

- file tempg.mat written ok.
- file tempz.mat written ok.
- file temp.dat written ok.
```

SSSTAT calculates C(**ε**) from observation weights and covariance matrices supplied in the PEST control file. Relativity of these is preserved while a factor is applied to all of these such that the objective function is equal to the sum of non-zero-weighted observations.

It is important to note that any regularisation observations or prior information equations are ignored. Hence the analysis carried out by SSSTAT is based purely on subspace concepts.

SSSTAT can obtain C(**k**) from a parameter uncertainty file. Alternatively, if instructed to do so, it can assume that C(**k**) is diagonal, and that its diagonal elements can be calculated from parameter bounds. Recall that the variance of a variable is the square root of its standard deviation. Like SUPCALC, SSSTAT assumes that the standard deviation of a parameter is equal to 0.3 of the distance between its lower and upper bounds. If a parameter is log-transformed then it is assumed to have a log-distribution, with the standard deviation of this distribution calculated in the same way from the logs of parameter bounds.

SSSTAT writes statistics (mainly the diagonal elements of matrices discussed above) to a file

of the user's choosing. It also records the **Z** matrix and **G** matrix in separate files. Each row of the former provides the means through which a scaled model output (to which field data is matched) is calculated from scaled model parameters. Each row of the latter provides the means through which an estimated scaled parameter is calculated from scaled observations. Individual rows of these matrices can be read using the MATROW utility. The contents of the file produced by this program can be listed in column form for greater readability using the MATTRANS utility. To make a bar chart, use a text editor that supports column cut and paste functionality to paste parameter or observation names alongside the numerical values of vector elements before importation into an appropriate spreadsheet or graphing package. Both names and vector element values are recorded in MATROW and MATTRANS output files.

When writing matrix diagonal elements to its output files, SSSTAT lists the names of the parameters and observations with which they are associated. It is important to note that parameter names are valid only if C(**k**) is diagonal. Similarly, observation names are valid only if C(**ε**) is diagonal. If C(**k**) is non-diagonal then at least some elements of **j** will actually be linear combinations of elements of **k**. If C(**ε**) is non-diagonal, then at least some elements of **f** will be linear combinations of elements of **h**. In spite of this, the same parameter and observation names are used for convenience.

3.40.4 Some Further Comments

SSSTAT provides a comprehensive set of statistics as they pertain to parameters and model outputs employed in the under-determined parameter estimation context. Further useful subspace information can be provided by the SUPOBSPAR1 utility.

Finally, it is worth noting that formulation of an objective function, and assignment of observation weights is as much an art as a science. The same data may be processed in a number of different ways with different weighting philosophies applied to each processing type. As Doherty and Welter (2010) explain, this can provide some defense against the deleterious effects of model structural noise on the inversion process. Doherty and Welter show that the covariance matrix of structural noise, though unknown, is almost certain to be singular. SSSTAT allows a user to experiment with different objective function formulations, and different weighting strategies. For any particular formulation/weighting strategy that he/she experiments with, SSSTAT allows a modeler to inspect the effects of this strategy on the dimensionality of the parameter solution space, estimability of individual parameters, and the information content of various observations or groups of observations.

3.40.5 References

Doherty, J. and Hunt, R.J., 2009. Two statistics for evaluating parameter identifiability and error reduction. *Journal of Hydrology*. 366, 119-127.

Doherty, J. and Welter, D., 2010, A short exploration of structural noise, *Water Resour. Res*., 46, W05525, doi:10.1029/2009WR008377.

Hill, M.C., and Tiedeman, C.R., 2007. Effective Groundwater Model Calibration and Analysis of Data, Sensitivities, Predictions, and Uncertainty. John Wiley and Sons, New York.

Moore, C. and Doherty, J., 2005. The role of the calibration process in reducing model predictive error. *Water Resour. Res.*. Vol 41, No 5. W05050.

4. Model Predictive Error andUncertainty Analysis

4.1 Concepts

4.1.1 General

As is explained in Moore and Doherty (2005), the parameter error covariance matrix C(**p**-**p̲**) of a calibrated model can be calculated using the formula:-

$$C(\mathbf{p} - \underline{\mathbf{p}}) = (\mathbf{I} - \mathbf{R})C(\mathbf{p})(\mathbf{I} - \mathbf{R})^t + \mathbf{G}C(\boldsymbol{\varepsilon})\mathbf{G}^t \quad (4.1)$$

where:-

p represents "true" model parameters (which we never know);

p̲ represents calibrated model parameters;

C(**p**) represents the covariance matrix of true parameters (often described by a variogram);

C(**ε**) represent the covariance matrix of measurement noise (mostly assumed to be a diagonal matrix);

R is the so-called "resolution matrix"; and

G is the matrix through which estimated parameter values (i.e. the elements of **p̲**) are calculated from measurements, referred to herein as the "parameter solution matrix", or more simply as the "**G** matrix".

Let *s* be a model prediction whose sensitivities to model parameters are encapsulated in the vector **y**. For a linear model, the "true" value of a model prediction is given by:-

$$s = \mathbf{y}^t\mathbf{p} \quad (4.2a)$$

while its model-calculated counterpart is:-

$$\underline{s} = \mathbf{y}^t\underline{\mathbf{p}} \quad (4.2b)$$

Model predictive error (which is never known) is given by:-

$$s - \underline{s} = \mathbf{y}^t(\mathbf{p}-\mathbf{p}) = \mathbf{y}^t(\mathbf{I} - \mathbf{R})\mathbf{p} - \mathbf{y}^t\mathbf{G}\boldsymbol{\varepsilon} \quad (4.3)$$

while model predictive error variance (i.e. the "variance of potential wrongness" of a model prediction) is given by:-

$$\sigma^2_{s-\underline{s}} = \mathbf{y}^t(\mathbf{I} - \mathbf{R})C(\mathbf{p})(\mathbf{I} - \mathbf{R})^t\mathbf{y} + \mathbf{y}^t\mathbf{G}C(\boldsymbol{\varepsilon})\mathbf{G}^t\mathbf{y} \quad (4.4)$$

Note that while derivation of these equations rests on an assumption of model linearity, they are nevertheless correct to a good approximation when applied to many non-linear models – good enough, for example, to be used in the ranking of different data acquisition strategies in terms of their comparative ability to reduce the potential wrongness of a key model prediction. Furthermore, their application can be extended to nonlinear analysis without too much difficulty; software to implement the nonlinear extension of this theory is presently under construction.

4.1.2 Calculation of the R and G Matrices

Formulas for **R** and **G** depend on the method used by PEST to solve the inverse problem. For an overdetermined system, for which the regularisation opportunities offered by truncated SVD, SVD-assist and Tikhonov schemes are not required, the resolution matrix **R** is simply **I**. However in many cases potentially unstable overdetermined problems are rescued from numerical instability by use of a high Marquardt lambda (which is a de-facto Tikhonov regularisation device). The Marquardt lambda is employed in the calculation of the **R** and **G** matrices in the utility software described below. However it must be noted that this is not a very good regularisation device, and in many cases can lead to loss of diagonal dominance of the resolution matrix. Hence, whether using one of the specialist regularisation devices offered by PEST, or not, attempts should be made to keep the Marquardt lambda low (or even zero, as is suggested when using truncated SVD as a regularisation mechanism).

Formulas used for calculation of **R** and **G** are now provided. Variables used in these formulas are as follows:-

X This is the Jacobian matrix, each row of which is the derivative of a particular model outcome for which there is a complementary field measurement, with respect to all adjustable parameters. Note that for SVD-assisted parameter estimation, **X** refers to derivatives taken with respect to super parameters. Note also that if a parameter is log transformed, pertinent elements of **X** pertain to the log of that parameter.

Z The **Z** matrix is used only in formulas pertaining to SVD-assisted parameter estimation. This contains derivatives of model outcomes with respect to base parameters (or their logs). More is said on this below.

λ The PEST-calculated Marquardt lambda.

T The matrix of Tikhonov regularisation constraints. These constraints are assumed to be of the form $\mathbf{T}\underline{\mathbf{p}} = \mathbf{0}$.

S The relative regularisation weight matrix (calculated from user-supplied regularisation weights and/or user-supplied regularisation covariance matrices).

β^2 The PEST-calculated regularisation weight factor.

h The set of observations on which the parameter estimation process is based. ε featured in equations 4.1, 4.3 and 4.4 is the "noise" or "measurement error" associated with these observations.

Q The measurement weight matrix (calculated from user-supplied measurement weights and user-supplied measurement covariance matrices).

V The matrix whose columns are orthogonal unit eigenvectors of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ as calculated through singular value decomposition undertaken either during every iteration of the parameter estimation process (when this is achieved through truncated SVD), or at the beginning of the parameter estimation process for determination of super parameters (if using SVD-assisted parameter estimation).

E A diagonal matrix whose elements are the eigenvalues of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ (arranged in decreasing order) determined through singular value decomposition.

$\mathbf{V}_1$ The first *k* columns of **V**, where *k* is the singular value truncation limit, or the number of super parameters employed in SVD-assisted parameter estimation.

$\mathbf{E}_1$ A diagonal matrix whose elements are the first *k* eigenvalues of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$.

The utility software described shortly through which **R** and **G** can be calculated employs the **X** matrix corresponding to the best parameter set achieved through the parameter estimation process. This is stored in file *case.jco* which, like *case.rsd* (see below) and *case.par* (the parameter value file) is updated by PEST whenever an improved parameter set is obtained. The **Z** matrix, however, is not updated through the parameter estimation process. The utility software documented below provides the user with the option of using the **Z** matrix computed during the pre-SVD-assist base parameter sensitivity run, or of using a new **Z** matrix computed using optimised parameters; if possible, it is better to use the latter.

Formulas through which **R** and **G** are calculated are now presented.

Overdetermined parameter estimation.

$$\mathbf{R} = (\mathbf{X}^t\mathbf{Q}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{X} \quad (4.5a)$$

$$\mathbf{G} = (\mathbf{X}^t\mathbf{Q}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q} \quad (4.5b)$$

Tikhonov Regularisation

$$\mathbf{R} = (\mathbf{X}^t\mathbf{Q}\mathbf{X} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{X} \quad (4.6a)$$

$$\mathbf{G} = (\mathbf{X}^t\mathbf{Q}\mathbf{X} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q} \quad (4.6b)$$

Singular Value Decomposition with Zero Marquardt Lambda

$$\mathbf{R} = \mathbf{V}_1\mathbf{V}_1^t \quad (4.7a)$$

$$\mathbf{G} = \mathbf{V}_1\mathbf{E}_1^{-1}\mathbf{V}_1^t\mathbf{X}^t\mathbf{Q} \quad (4.7b)$$

SVD-Assist

$$\mathbf{R} = \mathbf{V}_1(\mathbf{X}^t\mathbf{Q}\mathbf{X} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{Z} \quad (4.8a)$$

$$\mathbf{G} = \mathbf{V}_1(\mathbf{X}^t\mathbf{Q}\mathbf{X} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q} \quad (4.8b)$$

The following should be noted.

1. PEST allows various combinations of different regularisation schemes to be used in calculating optimised parameter values. For example a non-zero Marquardt lambda can be used with truncated SVD, truncated SVD can be used as a matrix equation solution scheme in SVD-assisted parameter estimation, SVD-assist can be implemented with or without Tikhonov regularisation, etc. All of these (and other) permutations can be accommodated in the software described below.
2. Where some parameters are log-transformed, the pertinent elements of the **R** and **G** matrices calculated through the above equations pertain to the logs of these parameters.
3. Where SVD-assisted parameter estimation is undertaken, the **R** and **G** matrices pertain to base parameters (or their logs), as used by the model – not the super parameters used by PEST in the SVD-assisted parameter estimation process.

4.1.3 Some Special Considerations

4.1.3.1 Regularisation Relationships

As mentioned above, where Tikhonov regularisation is employed it is assumed to be of the type:-

$$\mathbf{T\underline{p}} = \mathbf{0} \qquad (4.9)$$

In PEST, regularisation can be linear (supplied through prior information equations) or nonlinear (supplied as observations). In both cases they are identified as regularisation relationships through being assigned to an observation group whose name begins with "regul". However PEST also allows regularisation relationships of the following type to be supplied:-

$$\mathbf{Tp} = \mathbf{k} \qquad (4.10)$$

Calculation of the resolution matrix, as implemented in the utility software described below, cannot accommodate relationships of the type expressed by equation 4.10. Fortunately, in most cases, equation 4.10 is easily transformed to equation 4.9 by appropriate parameter re-definition.

4.1.3.2 Initial Parameter Values

When using truncated SVD or SVD-assisted parameter estimation, integrity of the predictive error variance analysis process requires that initial base parameter estimates (provided in the "parameter data" section of the PEST control file) correspond to most likely parameters according to a user's conception of parameter likelihood based on the current modelling context and the characteristics of the modelled area.

*4.1.3.3 The **Z** Matrix*

As mentioned above, the **Z** matrix appearing in equation 4.8a provides the sensitivities of model outputs for which there are corresponding field measurements to base parameters. In SVD-assisted parameter estimation these can far outnumber super parameters, and computation of the **Z** matrix can therefore be costly. Nevertheless, as described in the PEST manual, this matrix must be calculated (based on initial parameter values) prior to the undertaking of SVD-assisted parameter estimation, and so should be available for calculation of the resolution matrix upon completion of the SVD-assisted parameter estimation process. A better matrix to use in (4.8a) however is a **Z** matrix calculated on the basis of optimised parameter values. Thus, after an SVD-assisted PEST run is complete, the PARREP utility can be used to build a new base PEST control file using optimised base parameter values. NOPTMAX can be set to "-1" in this new file so that when PEST is run it terminates execution as soon as the Jacobian matrix is filled. The resulting "JCO" file will then hold the **Z** matrix of sensitivities, calculated on the basis of optimized parameter values.

4.2 Alterations to PEST

4.2.1 The IRES Variable

In addition to its normal suite of output files, PEST now writes a "resolution data file" named *case*.*rsd* where *case* is the filename base of the current PEST control file. If desired, writing of this file can be enabled or suppressed using a new variable (named "IRES") which should be supplied following the IEIG variable on the tenth line of the PEST control file. If IRES is

omitted, its value is assumed to be "1" if PEST is run in regularisation mode and/or if PEST's SVD or SVD-assist functionality is activated, thus ensuring that file *case.rsd* is written. However if IRES is set to zero, writing of *case.rsd* is suppressed. (It is automatically set to zero if PEST is run in predictive analysis mode.) The figure below shows the structure of the "control data" section of the PEST control file with the IRES variable included.

```
* control data
RSTFLE PESTMODE
NPAR NOBS NPARGP NPRIOR NOBSGP
NTPLFLE NINSFLE PRECIS DPOINT NUMCOM JACFILE MESSFILE
RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM
RELPARMAX FACPARMAX FACORIG
PHIREDSWH
NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR
ICOV ICOR IEIG IRES
```

Structure of the "control data" section of a PEST control file.

4.2.2 The "Resolution Data File"

The resolution data file *case.rsd* is a binary file whose contents cannot be read by the user. Instead it is used by the RESPROC utility described below for calculation of the **R** and **G** matrices of equations 4.5 to 4.8.

Upon commencement of execution, PEST deletes an existing resolution data file having the same filename base as that of the current PEST control file if such a file is found. This eliminates the possibility that an old file will be confused for a new one if PEST did not run long enough to produce this file, or if IRES was inadvertently set to zero.

PEST updates the resolution data file many times during the course of the parameter estimation process such that data contained within it always pertains to the best parameters achieved so far during that process; it is thus overwritten whenever the estimated parameter set is improved.

4.3 RESPROC

4.3.1 General

RESPROC stands for "RESolution data postPROCessor". It is run after completion of a PEST run; normally PEST will have been run in regularisation mode, or with SVDMODE set to 1, or with its SVD-assist functionality activated, or a combination of these. However RESPROC can also provide useful results after a PEST run in which traditional parameter estimation was implemented without the aid of any regularisation device (except the Marquardt lambda). In all cases a "resolution data file" (named *case.rsd* where *case* is the filename base of the PEST control file) must have been produced on that PEST run.

RESPROC's task is to write a file containing both the **R** and **G** matrices pertaining to the previous PEST run, which can then be used by other utility programs for calculation of parameter and predictive error variances. In order to save disk space this, too, is a binary file; however the **R** and **G** matrices can be rewritten in ASCII form if desired using the RESWRIT utility described below.

4.3.2 What RESPROC Does

RESPROC reads the following files, all associated with an existing PEST dataset characterised by the *case* filename base:-

1. the PEST control file (named *case.pst*);
2. the resolution data file (named *case.rsd*);
3. the Jacobian matrix file (named *case.jco*).

(Note that at any stage of the parameter estimation process the contents of the latter two files pertain to the best parameters achieved at that stage of the process.)

If the previous PEST run implemented SVD-assisted parameter estimation, the following files are also read by RESPROC:-

1. the base pest control file in which base parameters are defined (named *bcase.pst*);
2. the base Jacobian matrix file in which base parameter sensitivities are recorded (named *bcase.jco*);
3. optionally, an updated JCO file in which base sensitivities are recorded for optimised base parameter values.

Where many parameters and observations are involved in the parameter estimation process, RESPROC may take a while to run, for the matrices that must be manipulated in the formulation of **R** and **G** can be large. Fortunately, it need only be run once, for these matrices, once calculated, can then be used for computation of the error variance of a variety of model predictions.

As presently programmed there is a slight restriction on the use of RESPROC, which it is hoped will not limit its usefulness too much. RESPROC insists that no covariance matrix in lieu of observation weights be used for observation groups pertaining to measurements; however it will accept the use of a covariance matrix for an observation group containing regularisation information.

4.3.3 Running RESPROC

RESPROC is run using the command:-

```
resproc case outfile
```

where :-

case is the filename base of the PEST control file pertaining to a completed PEST run, and

outfile is the name of the RESPROC output file containing the **R** and **G** matrices (referred to herein as a "binary resolution matrix file").

When supplying *case* on the RESPROC command line, the "*.pst*" extension can be omitted; if so, RESPROC will supply this extension automatically. *outfile*, however, must have its extension supplied; if it is omitted, RESPROC will assume that this file has no extension.

As it executes, RESPROC writes its current activities to the screen. As discussed above, these activities involve manipulation of possibly large matrices. They also involves matrix inversion and possibly singular value decomposition (which will need to be undertaken twice if the previous parameter estimation was SVD-assisted and if truncated SVD was employed

for estimation of super parameters). Hence, as mentioned above, execution of RESPROC may take a while. When it has completed calculation of **R** and **G**, RESPROC stores them in its output binary resolution matrix file, records this fact to the screen, and ceases execution.

4.3.4 SVD-Assisted Parameter Estimation

Where the previous PEST run implemented SVD-assisted parameter estimation, RESPROC prompts the user for some extra information as follows:-

```
Select option for obtaining base parameter sensitivities:-
    enter "1" to use those in base Jacobian matrix file bcase.jco
    enter "2" to read from another JCO file
Enter your choice:
```

If your response to the above prompt is "2", RESPROC asks for the name of a JCO file. This file must cite the same parameters and observations (including prior information) as the base parameter PEST control file used in setup of the SVD-assisted run. This will be automatically ensured if the following steps are taken for preparation and implementation of SVD-assisted parameter estimation, and subsequent postprocessing.

1. A PEST case is set up involving base parameters, and optional Tikhonov regularisation constraints. Let us suppose that the PEST control file for this case is named *bcase.pst*.
2. Base parameter sensitivities are calculated for this base case. These will be stored in the base Jacobian matrix file *bcase.jco*.
3. A super parameter dataset is constructed using SVDAPREP; let the new PEST control file be named *case.pst*.
4. PEST is run; after this run is complete, optimised base parameters reside in file *bcase.bpa*.
5. PARREP is run to build a new base PEST control file, *bcase1.pst* in which optimized parameter values are employed as parameter initial values. The command is:-

   ```
   parrep bcase.bpa bcase.pst bcase1.pst
   ```

6. NOPTMAX is set to "-1" in *bcase1.pst*. Thus it calculates the Jacobian matrix (the **Z** matrix of equation 4.8a), and then terminates execution. This matrix is stored in file *bcase1.jco*.
7. When RESPROC is run, a "2" is supplied in response to the above prompt. *bcase1.jco* is then supplied as the name of the alternative base Jacobian matrix file.

Limited experience to date suggests that use of a **Z** matrix calculated on the basis of optimized base parameter values results in a better resolution matrix than use of the original **Z** matrix contained in file *bcase.jco* which was used for definition of super parameters.

4.4 RESWRIT

4.4.1 General

The **R** and **G** matrices written by RESPROC are not readable by the user. If it is desired that these matrices be subject to inspection and/or plotting, they should be converted to ASCII format. RESWRIT accomplishes this task.

4.4.2 Running RESWRIT

RESWRIT is run using the command:-

```
reswrit resprocfile matfile1 matfile2
```

where

resprocfile	is the name of an unformatted RESPROC output file;
matfile1	is a "matrix file" to which the resolution matrix will be written; and
matfile2	is a "matrix file" to which the **G** matrix will be written.

Note that an extension should be provided for each of these files, for RESWRIT employs no default extensions.

4.4.3 Format of Matrix Files

A matrix file holding a matrix with three rows and four columns is illustrated below.

```
   3   4   2
3.4423    23.323    2.3232   1.3232
5.4231    3.3124    4.4331   3.4442
7.4233    5.4432    7.5362   8.4232
* row names
apar1
apar2
apar3
* column names
aobs1
aobs2
aobs3
aobs4
```

An example of a matrix file.

The first line of a matrix file contains 3 integers. The first two indicate the number of rows and number of columns in the following matrix. The next integer (named ICODE) is a code, the role of which will be discussed shortly.

Following the header line is the matrix itself, in which entries are space-separated and wrapped to the next line if appropriate. Because the elements are read using FORTRAN list-directed input, other options are available as well for representation of the matrix. Thus, for example, the line:-

```
5*3.0, 3*2.4, 2*4.4
```

is equivalent to:-

```
3.0 3.0 3.0 3.0 3.0 2.4 2.4 2.4 4.4 4.4
```

Then, if ICODE is set to 2, follows the string "* row names". Following that are NROW names (of 20 characters or less in length), containing the names associated with rows of the matrix. NCOL column names follow in a similar format.

For a square matrix ICODE can be set to "1". This indicates that rows and columns are associated with the same names (as is the case for a resolution matrix). In this case the string "* row and column names" follows the matrix, and the pertinent names are listed on the NROW lines following that.

A special ICODE value is reserved for diagonal matrices. If NCOL is equal to NROW, then

ICODE may be set to "-1". In this case only the diagonal elements of the matrix need to be presented following the integer header line; these should be listed one to a line as illustrated in the following figure. Following that should be the string "* row and column names" (for if ICODE is set to "-1" it is assumed that these are the same), followed by the names themselves.

```
5  5  -1
4.5
4.5
2.4
7.53
5.32
* row and column names
par1
par2
par3
par4
par5
```

A matrix file containing a diagonal matrix.

4.5 PARAMERR

4.5.1 General

PARAMERR's task is to construct the "parameter error covariance matrix" C(**p**-**p**) of equation (4.1). It stores the two terms on the right side of equation 4.1 in separate files. This saves the user from having to re-compute both of these terms if an input required by only one of them (for example C(**p**) or C(**ε**)) is altered. It also allows the user to quantify the individual contribution to overall predictive error variance made by "uncaptured system heterogeneity" on the one hand (the first term), and measurement error on the other (the second term). The first diminishes as the level of fit between model outputs and field measurements increases, while the second term grows as better model-to-measurement fits are obtained; as Moore and Doherty (2005) point out, the optimal level of model-to-measurement misfit for any particular parameter estimation problem is that which minimises the variance of one or a number of key model predictions.

4.5.2 Uncertainty Files

4.5.2.1 Concepts

Calculation of C(**p**-**p**) requires that the user provide the covariance matrices C(**p**) and C(**ε**). These must each be specified through an "uncertainty file", an example of which is shown below.

```
# An example of an uncertainty file

START STANDARD_DEVIATION
  std_multiplier 3.0
  ro9  1.0
  ro10 1.0
  ro4  1.0
END STANDARD_DEVIATION

START COVARIANCE_MATRIX
  file "mat.dat"
  variance_multiplier 1e-2
END COVARIANCE_MATRIX

START PEST_CONTROL_FILE
  file "te st.pst"
  variance_multiplier 2.0e2
END PEST_CONTROL_FILE
```

Example of an uncertainty file.

The purpose of an uncertainty file is to allow the user a number of different options for characterising the uncertainty of a group of entities comprising a vector quantity (for example **p** and **ε**). Three such options are presently available, viz. a list of individual entity standard deviations, a covariance file, and entity weights listed in the "observation data" section of a PEST control file. A single such option can be used to specify the entirety of a covariance matrix, or different mechanisms can be used to characterise different parts of the total covariance matrix.

An uncertainty file is subdivided into blocks. Each such block implements one of the mechanisms of uncertainty characterisation described above. An uncertainty file can have as many blocks as desired. However the following rules must be observed.

1. Any one uncertainty file is used to characterize the uncertainty of either parameters or observations, but not both.
2. Parameters and observations cited in an uncertainty file, and the files cited therein, are matched by name to those featured in the current parameter estimation problem.
3. The uncertainty of an individual element of the overall **p** or **ε** vector can be characterised in only one way. Thus any particular element cannot be cited in the "observation data" section of a PEST control file cited in a particular uncertainty file if it is also cited in a STANDARD_DEVIATION block of that uncertainty file, or in a covariance matrix file cited in the COVARIANCE_MATRIX block of that same uncertainty file.
4. An uncertainty file, and files cited therein, can cite the names of parameters and observations that are not featured in the current parameter estimation problem; data pertaining to these surplus parameters and observations are simply ignored. However it must not omit any of the parameters or observations pertaining to the current parameter estimation problem.
5. If a parameter is log-transformed in the current parameter estimation problem, then specifications of variance, covariance or standard deviation provided in the uncertainty file must pertain to the log of the parameter. PARAMERR provides no checks for this, for it has no way of knowing the transformation status of a particular parameter; it is thus the user's responsibility to ensure that this protocol is observed.

6. An uncertainty file used for the characterisation of C(**p**) cannot include a PEST_CONTROL_FILE block, for PARAMERR reads only the "observation data" section of a cited PEST control file.

7. As presently programmed, an uncertainty file used for characterisation of C(**ε**) must not include a COVARIANCE_MATRIX block, for PARAMERR assumes that measurement noise is uncorrelated.

Each block of an uncertainty file must begin with a START line and finish with an END line as illustrated in the above figure; in both cases the type of block must be correctly characterised following the START and END designators. Within each block, data entry must follow the keyword protocol concept. Thus each line must comprise a keyword, followed by the value (numerical or text) associated with that keyword. Filenames must be surrounded by quotes if they contain spaces. With one exception (the *std_multiplier* keyword in the STANDARD_DEVIATION block), keywords within a block can be supplied in any order; some can be omitted if desired. Keywords and block names are case insensitive.

Blank lines can appear anywhere within an uncertainty file. So too can comment lines; these are recognised through the fact that their first character is "#".

Each of the blocks appearing in an uncertainty file is now discussed in detail.

4.6.2.2 The STANDARD_DEVIATION Block

In a STANDARD_DEVIATION block, entity names (individual parameters or observations) are listed one to a line followed by their standard deviations. As stated above, if a parameter is log-transformed in the parameter estimation process, then this standard deviation should pertain to the log (to base 10) of the parameter. Parameters/observations can be supplied in any order. Optionally a *std_multiplier* keyword can be supplied in the STANDARD_DEVIATION block; if so, it must be the first item in the block. All standard deviations supplied on ensuing lines are multiplied by this factor (the default value of which is 1.0).

Parameters/observations cited in a STANDARD_DEVIATION block are assumed to be uncorrelated with other parameters/observations. Thus off-diagonal elements of C(**p**) or C(**ε**) corresponding to these items are zero. Pertinent diagonal elements of C(**p**) and C(**ε**) are calculated by squaring the standard deviation (after multiplication by the *std_multiplier*).

4.5.2.3 The PEST_CONTROL_FILE Block

Only two keywords are permitted in this block, these being the *file* and *variance_multiplier* keywords; the latter is optional, the default value being 1.0.

The name of a PEST control file should follow the *file* keyword. PARAMERR reads the "observation data" section of this file. For any particular observation cited in this section that is also featured in the current parameter estimation problem, PARAMERR calculates its variance from the weight w cited in the PEST control file as $(1/w)^2$. This variance is then multiplied by the *variance_multiplier* before insertion into the appropriate diagonal element of C(**ε**). Corresponding off-diagonal elements of C(**ε**) are assumed to be zero.

4.5.2.3 The COVARIANCE_MATRIX Block

Where one or more covariance matrices are supplied for subgroups of **p** which show intra-parameter correlation, these matrices are included in the larger C(**p**) matrix calculated by

PARAMERR (together with variances calculated from parameter standard deviations supplied in one or more STANDARD_DEVIATION blocks). Optionally, all elements of such a user-supplied covariance matrix can be multiplied by a factor, this factor (for which the default value is 1.0) being supplied following the *variance_multiplier* keyword.

The format of a covariance matrix file must be identical to that described in Section 4.4.3. In particular, the first line of this file must include 3 integers, the first two of which (specifying the number of rows and columns in the matrix) must be identical. The third integer must be "1" or "-1". The matrix itself must follow this integer header line; elements within this matrix must be space-delimited; rows can be wrapped onto consecutive lines. This matrix must be followed by the string "row and column names". Following this must be the names of the parameters to which the matrix pertains.

The following should be noted.

1. A covariance matrix must be positive definite.
2. The order of rows and columns of the covariance matrix (which corresponds to the order of parameters represented by this matrix as listed below the "row and column names" header) is arbitrary. PARAMERR will re-arrange these rows so that they correspond to the order of adjustable parameters supplied in the PEST control file on which the current parameter estimation problem is based.
3. A user-supplied covariance matrix need not contain all of the parameters pertaining to the current parameter estimation problem, for it need contain only a subset of these. (Other parameters can be cited in other covariance matrix files and/or one or more STANDARD_DEVIATION blocks supplied in the same uncertainty file.) However it must not contain any parameters which do NOT pertain to the current parameter estimation problem (unless none of its parameters pertain to the current parameter estimation problem, in which case the matrix is ignored).
4. If a parameter is log-transformed, the variance and covariances pertaining to that parameter as supplied in a covariance matrix file must in fact pertain to the log of that parameter.

4.5.3 Using PARAMERR

PARAMERR receives information from keyboard input supplied by the user in response to a set of prompts. (It requires too much information to be supplied through command-line arguments.)

Upon commencement of execution, PARAMERR prompts:-

```
Enter name of RESPROC output file:
```

in response to which the name of the RESPROC-generated unformatted resolution matrix file should be supplied (this file also contains the **G** matrix). Next it asks:-

```
Options are as follows:-
  to compute (I-R)C(p)(I-R)'     - enter 1
  to compute GC(e)G'             - enter 2
  to compute both                - enter 3
Enter your choice:.
```

As mentioned above, there will be occasions when only one of the two terms on the right side of equation 4.1 will require computation. However if both are required, enter 3. If this is done, PARAMERR's next prompts will be:-

```
 Enter name of parameter uncertainty file:
 Enter name for covariance matrix output file:

 Enter name of observation uncertainty file:
 Enter name for covariance matrix output file:
```

As discussed above, the uncertainty file in each case must be prepared by the user. The covariance matrix output file written by PARAMERR in each case is the respective term of equation (4.1), that is $(\mathbf{I} - \mathbf{R})\mathrm{C}(\mathbf{p})(\mathbf{I} - \mathbf{R})^t$ in the first case and $\mathbf{G}\mathrm{C}(\boldsymbol{\varepsilon})\mathbf{G}^t$ in the second case. These are written in standard matrix file format, following the protocol discussed in Section 4.4.3. Each of these matrices is square with dimensions $n \times n$ where n is the number of adjustable parameters pertaining to the current parameter estimation problem.

4.6 PREDERR

4.6.1 General

PREDERR is similar in many respects to PARAMERR. Like PARAMERR it reads a parameter uncertainty file and an observation uncertainty file; see documentation of PARAMERR for the format of these files. However rather than calculating and storing the two components of $\mathrm{C}(\mathbf{p}–\underline{\mathbf{p}})$, it directly calculates the error variance of a user-specified prediction. Thus calculation of $\mathrm{C}(\mathbf{p}–\underline{\mathbf{p}})$ is bypassed. This can result in enormous savings in calculation of predictive error variance. Thus unless calculation of $\mathrm{C}(\mathbf{p}\text{-}\underline{\mathbf{p}})$ is specifically required for a certain application, use of PREDERR is preferred over use of PARAMERR followed by JROW2VEC and MATQUAD (see the next section) for the calculation of the error variance of a certain prediction.

4.6.2 Using PREDERR

Many of PREDERR's prompts are similar to those of PARAMERR. However, unlike PARAMERR (but like JROW2MAT and JROW2VEC), PREDERR requires the name of a Jacobian matrix file, as well as the name of a particular observation for which parameter sensitivities are recorded in this file. This "observation" is treated as a "prediction" for the purpose of error variance calculation as undertaken by PREDERR using equation 4.4. That is, the parameter sensitivities extracted from the user-specified row of the Jacobian matrix constitute the **y** vector of this equation.

PREDERR's screen display, including its prompts, are as follows; typical responses are shown highlighted.

```
Enter name of RESPROC output file: modela.rpo

Enter name of parameter uncertainty file: param1.unc
Enter name of observation uncertainty file: observ1.unc

Enter name of Jacobian matrix file: model1.jco
Enter name of prediction featured in this file: ptime

- reading RESPROC output file modela.rpo...
- file modela.rpo read ok.

- reading Jacobian matrix file model1.jco...
- Jacobian matrix file model1.jco read ok.

- reading parameter uncertainty data...
```

```
- parameter uncertainty data read ok.

- calculating I-R contribution to predictive error variance...
- I-R term calculated ok.

- reading observation uncertainty data...
- reading PEST control file temp.pst...
- file temp.pst read ok.
- observation uncertainty data read ok.

- calculating G contribution to predictive error variance...
- G term calculated ok.
```

The following should be noted.

1. The Jacobian matrix file will have been written by PEST. It is an unformatted file with the name *case.jco* where *case* is the filename base of the corresponding PEST control file.
2. The cited Jacobian matrix file must contain an observation named as the "prediction" in the pertinent PREDERR prompt. Many more observations than this can be cited in the Jacobian matrix file; all others are ignored.
3. Each parameter cited in the RESPROC output file must also be cited in the Jacobian file. If more parameters than this are cited in the Jacobian matrix file, they are ignored. If fewer parameters are cited, PREDERR ceases execution with an appropriate error message. Parameters need not be arranged in the same order in the Jacobian matrix file as they are in the RESPROC output file (and hence in the PEST control file on which the RESPROC output file is based).

PREDERR quickly calculates the contribution to predictive error variance made by both terms of equation 4.4 (which it refers to as the "I-R term" and the "G term"). It writes the outcomes of its calculations to the screen, an example of which is depicted below.

```
*********** COMPONENTS OF PREDICTIVE ERROR VARIANCE  ***********

      I-R component of predictive error variance =   2881711.
      G   component of predictive error variance =   5342.439
      Total            predictive error variance =   2887054.
      Predictive error standard deviation        =   1699.133

 ************************************************************
```

4.7 PREDERR1

Operation of PREDERR1 is very similar to that of PREDERR, the only difference being that PREDERR1 does not prompt for the name of a RESPROC output file. Instead it prompts individually for the names of files which hold the resolution matrix on the one hand and the G matrix on the other. These files must be in matrix file format as described in the following section.

Note the following.

1. Row names provide in the resolution and G matrix files must be identical to each other.

2. The resolution matrix must have identical row and column names. These names must be the same as parameter names involved in the current predictive error variance analysis problem.
3. Column names cited in the G matrix must pertain to observations employed in the current predictive error variance analysis problem.

4.8 PREDERR2

PREDERR2 is a modification of PREDERR1. Unlike PREDERR1, which calculates error variance for only one prediction whose sensitivities are contained within a Jacobian matrix file, PREDERR2 can calculate variances for multipliple predictions featured in this file. Like PREDERR1 it prompts for a resolultion matrix file, a G matrix file, a parameter uncertainty file, an observation uncertainty file and a Jacobian matrix file. However it also prompts for the name of a "prediction list file". This must contain a list of predictions, one to a line, for which error variances are to be calculated; each such prediction should be featured in the Jacobian matrix file.

PREDERR2 also prompts for the name of a "prediction error variance output file". Part of this file is featured below.

Prediction	Variance_1	Variance_2	Total	Standard_Dev
ar1	0.4282991	4.7418460E-02	0.4757175	0.6897228
ar2	0.3441107	6.0151659E-02	0.4042623	0.6358163
ar3	0.2486108	8.1392663E-02	0.3300034	0.5744593
ar4	0.1724225	0.1177954	0.2902179	0.5387187
ar5	0.1363500	0.1767298	0.3130798	0.5595354
ar6	0.1115423	0.2602005	0.3717428	0.6097071

Part of a PREDERR2 output file.

The "variance_1" and "variance_2" terms are the I-R and "G" terms of the predictive error variance equation. Thus these terms provide the contributions to predictive error variance arising from "uncaptured heterogeneity" and measurement noise respectively. The "total" is the sum of these and the "standard_dev" is the square root of that.

4.9 PREDERR3

PREDERR3 is almost identical to PREDERR2. However instead of reading the resolution and "G" matrices from separte files, it reads both of these from an unformatted RESPROC output file.

4.10 PREDVAR1

4.10.1 General

PREDVAR1 analyses "notional predictive error variance" without requiring that a model be actually calibrated to do so. Equation 4.4 is employed on the assumption that the notional calibration exercise is completed using singular value decomposition (SVD).

PREDVAR1 can be extremely useful in achieving at least the following objectives.

1. Before a model is even calibrated, a rapid assessment can be made of the reduction in predictive variance that can be achieved through the calibration process, given the number and type of measurements comprising the calibration dataset, the noise C(**ε**)

associated with these measurements, and the inherent variability C(**p**) of model parameters that accrues from environmental system complexity and/or heterogeneity. By comparing post-calibration predictive variance with pre-calibration predictive variance calculated as

$$s = \mathbf{y}^t C(\mathbf{p})\mathbf{y} \qquad (4.11)$$

a rapid assessment can be made of the worth of undertaking a possibly expensive and time-consuming calibration exercise before that exercise is actually embarked upon.

2. Because the calibration exercise simulated by PREDVAR1 is notional rather than actual, the reduction in uncertainty achieved through including one or a number of hypothetical extra observations in the calibration dataset can be rapidly assessed. This can form a sound scientific basis of optimisation of data acquisition, based on the premise that the worth of acquiring a certain type of data over that of acquiring another type of data is greater if its acquisition results in a greater reduction in predictive error variance. Furthermore, because equation 4.4 is based on a full characterization of uncertainty (including the effects of system complexity which is beyond the ability of the calibration process to capture), this method of assessing the worth of gathering extra data is superior to similar statistics based on over-determined estimation of parameters pertaining to over-simplified models which, under most circumstances of environmental model calibration, are almost meaningless.
3. Similarly, by varying the terms of C(**p**) in accordance with an improvement in direct knowledge of system properties that may be gained through direct system measurement, an assessment of the worth of these measurements in reducing the uncertainties of one or more predictions can be made. Various data acquisition strategies of this type can then be ranked, and/or these strategies can be compared with the benefits of acquiring extra information on system states (see the above point) to use in a future calibration exercise.
4. By setting certain components, or groups of components, of C(**p**) to zero and by employing equation 4.4 and/or 4.11 in conjunction with this revised C(**p**), the contribution to the uncertainty of a specific model prediction made by different parameter types can be estimated. Where knowledge of the values of those parameter types cannot be enhanced by direct measurement, this may allow the user to determine the "irreducible level of uncertainty" associated with key model predictions.
5. By re-calculating error variance using equation 4.4 for different numbers of singular values at which the SVD process is truncated, the minimum in the curve of number of singular values versus model predictive error variance can be ascertained; see Moore and Doherty (2005). This can serve as a useful guide to the number of super parameters to employ in SVD-assisted parameter estimation.

As mentioned above, PREDVAR1 calculates predictive error variance on the assumption that regularisation of the notional calibration exercise is implemented using truncated SVD. Thus equations 4.7a and 4.7b are employed in the calculation of **R** and **G**. It is further assumed that the user supplies a set of measurement weights for this calibration exercise that satisfies the equation:-

$$C(\boldsymbol{\varepsilon}) = \sigma_r^2 \mathbf{Q} \qquad (4.12)$$

where **Q** is the weight matrix. Use of this relationship further simplifies (and expedites)

calculation of $\sigma^2_{s-\underline{s}}$ using equation 4.4. Note that one or more measurement covariance matrices can be used instead of weights as discussed below.

4.10.2 Using PREDVAR1

PREDVAR1 requires too many inputs for these to be supplied through its command line. So it prompts the user for information, which must be supplied in response to these prompts. PREDVAR1 commences execution with the prompt:-

```
Enter name of PEST control file:
```

Supply the name of a PEST control file. It is assumed that a complementary Jacobian matrix file ("JCO file") is available for this PEST control file. Thus it is assumed that PEST has been run for one iteration (or that NOPTMAX has been set to -1 on its previous run) so that derivatives have been calculated for all model outputs with respect to all adjustable model parameters. (Predictive sensitivities may also have been calculated – see below).

Next PREDVAR1 prompts:

```
Enter observation reference variance:
```

PREDVAR1 computes C(**ε**) of equation 4.4 by assuming that the inverse of each weight contained in the PEST control file is proportional to the standard deviation of the observation to which it is assigned. It is assumed that the proportionality constant is the same for all weights, and is equal to the square root of the reference variance depicted in equation 4.12. That is, the squares of weights are assumed to comprise the inverse of the diagonal of the **Q** matrix of equation 4.12. If desired the PEST control file can feature one of more observation covariance matrices as well, these being used instead of weights for one or more observation groups. The same reference variance must apply to these as well; that is, the elements of these matrices are multiplied by σ_r^2 of equation 4.12 in calculating the appropriate submatrix of C(**ε**).

PREDVAR1 next prompts for the name of a parameter uncertainty file. The prompt is:-

```
Enter name of parameter uncertainty file:
```

The contents of this file define C(**p**); see documentation of program PARAMERR for details. The uncertainties associated with elements, or groups of elements, of the parameter vector **p** can be defined in the parameter uncertainty file using standard deviations or covariance submatrices (possibly based on geostatistical characterizations of the subsurface). The following should be noted.

- If a parameter is log-transformed in the PEST control file, the elements of C(**p**) associated with that parameter must pertain to the log of that parameter.
- As is the protocol for a parameter uncertainty file, this file can contain information pertaining to more parameters than those which are demarcated as adjustable in the PEST control file. Unused parameters are simply ignored.
- If one parameter is tied to another parameter in the PEST control file, then the parent parameter is in fact a "composite parameter". Its statistical properties as supplied in C(**p**) should reflect this.

Next PREDVAR1 prompts:-

```
Enter name of predictive sensitivity matrix file:
```

This file must contain a single column matrix (in matrix file format as documented in Section

4.4.3) which contains the sensitivity of a prediction of interest to every adjustable parameter cited in the PEST control file. That is, it must contain the vector **y** of equation 4.4. The following should be noted.

- If a parameter is log transformed in the PEST control file, then parameter sensitivities contained in **y** must pertain to the log of the parameter.
- If a parameter is tied to another parameter in the PEST control file, then the sensitivity with respect to the parent parameter as contained in **y** must reflect the fact that it is parent to another parameter.
- Parameters do not need to be arranged in the same order in the predictive sensitivity matrix file as they are in the PEST control file. PREDVAR1 links parameters by name, and re-arranges them if necessary. Similarly, if the predictive sensitivity matrix file cites more parameters than are adjustable in the PEST control file, the excess parameters are simply ignored.

The easiest way to make a predictive sensitivity matrix file is as follows.

- When PEST is run in order to calculate observation sensitivities, design the model to run in such a way that it makes one or more predictions, as well as calculating outputs that correspond to historical observations listed in the PEST control file for calibration purposes. This may require that the model be run not once, but twice, based on different inputs, through a batch file serving as "the model" as seen by PEST.
- List these predictions in the PEST control file as additional "observations". However give them a weight of zero.
- If this is done, the JCO file produced as an outcome of the PEST run will contain the sensitivities of these predictions to all adjustable parameters. A predictive sensitivity file can be constructed for each such prediction using the JROW2VEC utility discussed in the next section.

Next PREDVAR1 issues a series of prompts as follows:-

```
Enter no. of singular values before truncation [<Enter> if no more]:
```

Enter a number between (and including) zero and the number of adjustable parameters employed in the simulation. Be aware, however, that if there are more adjustable parameters than observations, the number of non-zero singular values of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ will be only as large as the number of observations. So do not enter a number higher than this. PREDVAR1 will inform you if this number is too high and thus results in a zero singular value; in fact it will cease execution with an error message under these circumstances. So when entering singular values in response to the above series of prompts it is best to supply them in increasing order.

If truncation takes place at zero singular values, this is equivalent to not calibrating the model at all. Hence predictive error variance is calculated using equation 4.11. As the number of singular values prior to truncation increases, the first term of equation 4.4 will normally fall. In fact it will fall monotonically if C(**p**) is equal to $\sigma^2_p\mathbf{I}$ where **I** is the identity matrix and σ^2_p is an appropriate "parameter reference variance". In other cases, however, there may be interruptions in the fall (and sometimes local rises) with increasing singular value, indicating that calibration can actually *increase* the error variance of a prediction if C(**p**) contains variances of very different magnitude, and/or indicates a high level of parameter correlation. The former problem can be rectified by appropriate parameter scaling; this should be done as

a matter of course when calibrating a model in order to prevent this problem.

The second term of equation 4.4 should rise as the number of singular values prior to truncation increases. At some number of singular values there is normally an optimum, where predictive error variance is least.

PREDVAR1 finally prompts for the names of its two output files:-

```
Enter name for predictive error variance output file:
Enter name for SVD file [<Enter> to read an old one]:
```

The first file lists the contributions to predictive variance made by the first and second terms of equation 4.4 for each requested singular value. It also lists the total predictive error variance, together with the predictive standard deviation (square root of this). If singular values are arranged in increasing order, this file can serve as an input to a plotting/graphing program. The second file is similar to that produced by PEST when implementing singular value decomposition as an inversion device. It lists singular values and respective eigenvalues of $\mathbf{X}^t\mathbf{QX}$, calculated as an outcome of the SVD process. If the option it taken to read an old file, PREDVAR1 prompts for the name of this file. For large problems involving many parameters, making use of SVD data generated on a previous PREDVAR1 run can save a considerable amount of time. It is important to note, however, that the results of a previous singular value decomposition of $\mathbf{X}^t\mathbf{QX}$ are only appropriate for a current PREDVAR1 run if the PEST control file (and any covariance matrices cited therein) is the same for this run as it was for the run on which the SVD file was originally saved; however parameter uncertainty data can change between these two runs. Note also, that there my be tiny differences between variances calculated on the basis of stored SVD data and that calculated internally because of the slight loss of precision incurred through ASCII file storage.

It is also important to note that if the PEST control file read by PREDVAR1 instructs PEST to run in regularisation mode, regularisation observations and prior information equations are ignored in formulating $\mathbf{X}$ and hence in computation of predictive error variance. Thus it is assumed that the only type or regularisation undertaken is that pertaining to truncated SVD, with the truncation limit set at those singular values supplied by the user.

4.11 PREDVAR2

4.11.1 General

Operation of PREDVAR2 is based on the same principals as that of PREDVAR1. However instead of calculating the dependence of predictive error variance on the number of singular values employed in the truncated SVD process through which $\mathbf{R}$ and $\mathbf{G}$ of equation 4.4 are evaluated, it allows calculation of predictive error variance based on a sequence of different parameter uncertainty files, all for the same singular value truncation limit. This can be very useful for at least the following purposes.

1. If, in each of these files, a particular parameter, or parameter type, is assigned zero variance (and is therefore assumed to be perfectly known), the reduction in error variance accrued from knowing that parameter or parameter type, is a measure of the contribution made to the variance of the prediction by that parameter type. This analysis can be undertaken at an optimum number of singular values as determined by prior use of PREDVAR1. It can also be undertaken at zero singular values, to determine pre-calibration contribution of different parameters and/or parameter types

to predictive error variance. The change between pre- and post-calibration predictive error variance is a measure of the effect that the calibration process has in reducing the effect of that parameter or parameter type on the error variance of the studied prediction.

2. Where a parameter is spatially variable, and where this variability its characterized within a model using a mechanism such as pilot points, contribution to predictive error variance made by different parameters can be contoured. This presents a graphic means by which the worth of making direct property measurements in different parts of the model domain, as a means of reducing the uncertainty of a particular prediction, can be assessed. The variance of the "measured" prediction does not need to be reduced to zero in this process; it can simply be diminished, thus reflecting the fact that direct "measurements" of system properties have their own sources of uncertainty.

4.11.2 Using PREDVAR2

Like PREDVAR1, PREDVAR2 commences execution with the prompt:-

```
Enter name of PEST control file:
```

It is assumed that a corresponding JCO file is present (PREDVAR2 will cease execution with an error message if it is not). It is important to note that, as for PREDVAR1, the model need not be calibrated; hence values employed in the PEST control file (and those on which derivatives recorded in the JCO file are based) can be best parameter estimates originating from outside the calibration process. A JCO file pertaining to these parameter estimates can be obtained by running PEST on the basis of this PEST control file with NOPTMAX set to 1 or -1. In some circumstances it may be convenient to write this JCO file on the basis of a JCO file written on a previous PEST run using the JCO2JCO utility.

As for PREDVAR1, it is assumed that the inverse squares of weights contained in the cited PEST control file are proportional to measurement variances, and that any measurement covariance matrices cited in the PEST control file are related to true measurement covariances using the same proportionality constant; see equation 4.12. PREDVAR2 then asks for the value of this constant, that is for the reference variance. The prompt is:-

```
Enter observation reference variance:
```

Next PREDVAR2 prompts:-

```
Enter name of parameter uncertainty file listing file:
```

A parameter uncertainty file listing file must list the names of parameter uncertainty files, one after the other; see the example below.

```
# Horizontal K's set to zero
param1.unc
# Vertical K's set to zero
param2.unc
# Sy's set to zero
param3.unc
```

Part of a parameter uncertainty file listing file.

Any line within a "parameter uncertainty file listing file" that begins with the "#" character, or is blank, is ignored. Files cited within this file must be parameter uncertainty files (use quotes to surround filenames containing a blank), the format of which is described in the documentation to program PARAMERR. There is no limit to the number of files which can

be so listed. Each one of these files allows construction of a C(**p**) matrix as required for use in equations 4.4 and 4.11.

Next PREDVAR2 asks for the name of a predictive sensitivity matrix file. This contains the **y** vector of equations 4.4 and 4.11. This matrix can be built using a text editor if desired. However in most cases it is easier to extract it from a JCO file using the JROW2VEC utility. PREDVAR2's next prompt is:-

```
Enter number of singular values before truncation:-
```

Enter a number between 0 and the number of adjustable parameters cited in the PEST control file; on most occasions the number of singular values should correspond to the optimum for the prediction of interest identified using PREDVAR1.

PREDVAR2 next prompts for the name of its principal output file:-

```
Enter name for predictive error variance output file:
```

For each parameter uncertainty file listed in the parameter uncertainty file listing file, the PREDVAR2-generated variance output file will contain predictive error variance components corresponding to the first and second terms of equation 4.4, as well as the total predictive error variance; it will also list the square root of the later as the total predictive standard deviation.

Finally PREDVAR2 prompts:-

```
Enter name for SVD file [<Enter> to read an old one]:
```

PREDVAR2 records the outcomes of singular value decomposition of the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix in an ASCII file. This may be useful for its own sake (in order that the user may be aware of the linear parameter combinations defining orthogonal directions in parameter space which would correspond to super parameters if SVD-assist were to be employed in the calibration process). However if this file is read in a subsequent PREDVAR2 run (instead of re-calculating the singular value decomposition of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$), considerable savings in computation time can be achieved where large numbers of adjustable parameters are cited in the PEST control file (with slight loss or precision through having to read these results from an ASCII file). However for the SVD file to be valid for the current run, no alterations should have been made to the PEST control file on which the current PREDVAR2 run is based, between this run and its previous one. If the user presses the <Enter> key in response to the above prompt, signalling that such an old SVD file should be read, PREDVAR2 then prompts for the name of this file.

After all of the above questions have been answered, PREDVAR2 undertakes its calculations, recording the progress of its work to the screen. Its output file can be inspected at any time during these calculations.

4.12 PREDVAR3

4.12.1 General

PREDVAR3 bares some resemblance to PREDVAR2 in that it is employed for calculation of the contribution of different parameter types, or parameter groups, to the potential error of a prediction of interest. However the methodology underpinning its use is more general than that of PREDVAR2. Hence it is more suitable for use in contexts where predictive error is shared between both of the terms of equation 4.4. PREDVAR2, on the other hand, is suited to those contexts where the first term of equation 4.4 dominates predictive error variance, and

where the second term is small or absent (as it is when an assessment is being made of pre-calibration predictive error variance).

As for PREDVAR2 and PREDVAR1, use of PREDVAR3 does not depend on the existence of a calibrated model. All that is required is that sensitivities of model-generated measurement-equivalents to all parameters employed by the model be available where the model is run under calibration conditions, and that the sensitivities of a model prediction of interest to all parameters employed by the model also be available.

There are a number of ways in which the "contribution to total predictive error variance" by a particular parameter, or parameter type/group can be defined. One way in which such a quantity can be calculated is through undertaking the following exercise.

1. Use PREDVAR1 (possibly in conjunction with SCALEPAR in order to scale parameters by their inherent variabilities) to determine the minimum predictive error variance achievable for a certain prediction. This is accomplished by plotting predictive error variance vs. number of singular values, and identifying the minimum of this curve.
2. Build a new PEST control file (corresponding to calibration conditions, and possibly predictive conditions for the sake of obtaining predictive sensitivities) in which parameters comprising a particular type or group are held fixed, thus implying that they are notionally perfectly known by the modeller.
3. Obtain the Jacobian matrix pertaining to this new PEST control file. This is normally easily obtained from the Jacobian matrix corresponding to the original PEST control file (in which no parameters are fixed) using the JCO2JCO utility.
4. Run PREDVAR1 on the basis of this new PEST control file and complimentary Jacobian matrix file to calculate minimum predictive error variance. The difference between this predictive error variance and the original predictive error variance based on a full parameter set is a measure of the amount by which the attainment of perfect knowledge of the newly fixed parameters would reduce the error variance of the prediction of interest. It can thus be considered to quantify the "contribution" made by this parameter type to the error variance of the prediction of interest.

PREDVAR3 carries out the same operations as listed above, without the need for production of new PEST control files and Jacobian files, and without the need to run PREDVAR1 repeatedly. However because of its similarity to PREDVAR1, many of its inputs resemble that of this program.

4.12.2 Using PREDVAR3

PREDVAR3's first four prompts are the same as those of PREDVAR1. They are repeated below:-

```
Enter name of PEST control file:
Enter observation reference variance:

Enter name of parameter uncertainty file:
Enter name of predictive sensitivity matrix file:
```

See documentation of PREDVAR1 for details of how to respond to these prompts. (Note that use of PREDVAR3, like that of PREDVAR1, assumes that observation weights contained in the PEST control file are inversely proportional to measurement uncertainties, and/or that one or a number of measurement covariance matrices are supplied whose elements are related to

measurement noise by the same constant of proportionality, this being the reference variance referred to above; see documentation of PREDVAR1 for further details.)

Next PREDVAR3 prompts for a "singular value list" file. That is:-

```
Enter name of singular value list file:
```

This is a file containing singular values written one to a line. For every parameter type nominated by the user (see below), PREDVAR3 calculates the predictive error variance corresponding to all of these singular values. It then selects the minimum of these variances and records that on its output file. It is important to note that if a model has many parameters, then use of many singular values can lengthen PREDVAR3 execution time a great deal. Normally the user will know roughly where the minimum will lie, and can properly span this space. If the space is wide, then perhaps employing a singular value set with an increment of two or three, rather than unity, may also help. If it is found that a minimum error variance exists at either end of the supplied sequence of singular values, this is evidence that a large enough range of singular values may not have been spanned. PREDVAR3 lists the singular value corresponding to the minimum predictive error variance for each parameter type in its output file; see below.

Next PREDVAR3 prompts:-

```
Enter name of parameter type list file:
```

The format of such a file is demonstrated below.

```
* parameter type "none"
* parameter type ro
ro1
ro2
ro3
ro4
ro5
* parameter type thickness
th1
th2
th3
th4
* parameter type "bot_elev"
belev1
belev2
etc
```

Format of a "parameter type list" file.

Normally the parameter type list file will be prepared by extracting a list of parameters from the PEST control file for the current case and simply inserting parameter type identifiers within this list as shown above. Each set of parameters appearing between two such identifiers (or between an identifier and the end of the file) comprises a "parameter type", with the name of this type being defined after the "* parameter type" string which identifies the beginning of the listing for that type.

Note the following.

1. The list of parameters comprising a certain type can be empty. In this case no parameters are frozen and the predictive error variance is equivalent to that which would be computed by PREDVAR1 on the basis of all model parameters.

2. A given parameter can belong to more than one type if desired.
3. There is no limit to the number of parameter types which can be represented in a parameter type file.
4. The name of a parameter type should be 12 characters or less in length. It can optionally be surrounded by quotes.
5. If the overall C(**p**) matrix for the current case indicates that correlation exists between certain parameters (that is, if C(**p**) is not a diagonal matrix), then correlated parameters must belong to the same parameter type, and must thus be all frozen together or not at all. (Proper calculation of separate predictive error variance contributions by parameters which are correlated requires that the C(**p**) matrix be conditioned as a result of the attainment of perfect knowledge pertaining to the notionally fixed parameters. Where a fixed parameter is not correlated with any other parameter, such conditioning of C(**p**) is not necessary.)

For each parameter type identified in the parameter type listing file, PREDVAR3 computes the predictive error variance for all singular values nominated in the singular value list file. It then records the minimum such variance to its output file, an example of which is shown below. Also recorded are the first and second predictive error terms corresponding to the minimized predictive error variance.

```
Name of prediction = "ar10"

Fixed_params      First_term      Second_term     Minimized_error_variance  Sing_vals_at_min
none               4.4625057E-02   0.1285734       0.1731985                    4
top                4.0642764E-02   0.1438543       0.1844970                    2
bottom             1.9366123E-02   9.0539228E-03   2.8420046E-02                2
middle             3.4950000E-02   5.6072226E-02   9.1022226E-02                4
```

Part of a PREDVAR3 output file.

A PREDVAR3 output file is easily imported into a spreadsheet program such as EXCEL for undertaking the subtractions necessary to obtain contributions made by different parameter types to the variance of a certain prediction.

4.13 PREDVAR4

PREDVAR4's purpose and use are identical to those of PREDVAR3. However it offers a slightly higher level of functionality than PREDVAR3.

If certain parameters are frozen for the purpose of calculating their contribution to predictive error variance, and if those parameters are correlated with non-frozen parameters, the C(**p**) matrix of the latter is conditioned by the fact that the former are now notionally perfectly known, and hence have no uncertainty associated with them. Use of the conditioned C(**p**) matrix in predictive error variance computation results in an altered error variance, reflecting the fact that notional perfect knowledge of the frozen parameters has implications for knowledge of the "innate variability" of the non-frozen parameters as well.

Let the parameter vector **p** be partinioned into two separate parts, each pertaining to a separate group of parameters. Thus:-

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix}$$

C(**p**) is then correspondingly partitioned as:-

$$C(\mathbf{p}) = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix}$$

Suppose further that the elements of $\mathbf{p}_2$ are known. Then, if there is correlation between at least some members of $\mathbf{p}_2$ and some members of $\mathbf{p}_1$ (resulting in non-null $\mathbf{C}_{12}$ and $\mathbf{C}_{21}$ submatrices), the conditioned $\mathbf{C}_{11}$matrix $\mathbf{C}'_{11}$ is calculable as:-

$$\mathbf{C}'_{11} = \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}^{-1}{}_{22}\mathbf{C}_{21}$$

PREDVAR4 uses $\mathbf{C}'_{11}$ in calculation of the null space contribution to predictive error variance. (PREDVAR3, on the other hand, does not allow correlation to exist between frozen and non-frozen parameters when evaluating contributions of different parameter types to predictive error variance.)

4.14 PREDVAR5

4.14.1 General

PREDVAR5 is used for analysing the effect of individual observations, or groups of observations, on the error variance of a particular prediction. It has two modes of operation. These are as follows.

1. It allows ranking of the relative worth of present observations by calculating predictive error variance with selected ones, or groups, of these observations removed from the existing calibration dataset.
2. It allows ranking of the relative worth of new observations by calculating predictive variance with selected ones, or groups, of these new observations added to the existing calibration dataset.

As for PREDVAR3 and PREDVAR4 (which work with parameter contributions to predictive error variance instead of observation contributions), PREDVAR5 allows observations to be grouped into "types" for addition to, or subtraction from, the existing calibration dataset. The format of an "observation type list file" is very similar to that of the "parameter type list file" used by PREDVAR3 and PREDVAR4. An example of such a file follows.

```
* observation type "base"
ar8
ar9
..
ar34
* observation type "none"
* observation type "type1"
ar5
ar3
ar4
* observation type "flows"
flow1
flow2
* observation type "heads"
head1
head2
etc
```

Format of an "observation type list" file.

The PREDVAR5 observation addition and subtraction options will now be discussed in more detail.

4.14.1.1 Subtracting Observations

This is the easier of the two options to implement. In this case each set of observations belonging to the different observation types nominated in the observation type list file is, in turn, removed from an existing PEST input dataset as set out in an existing PEST control file; this is effected by setting all of the pertinent observations weights to zero. Not only observations can be removed, but prior information equations as well. However it is important to note that regularisation observations and prior information equations are ignored when PREDVAR5 (and any of the other PREDVAR programs) calculate predictive error variance. If the PEST control file whose name is provided to PREDVAR5 instructs PEST to run in regularisation mode, and if an observation or prior information equation is cited in an observation type list file which belongs to a regularisation group, PREDVAR5 will cease execution with an error message.

Naturally, each observation cited in an observation type list file must be cited in the PEST control file which defines the current calibration exercise.

An observation type can have no members; often the first nominated observation type will have no members. Thus PREDVAR5 will calculate predictive error variance with no observations subtracted from the current PEST dataset. The contributions that other nominated types of observation make to the accuracy of a certain prediction can then be evaluated by subtracting the error variance of that prediction with the observation type included, from that calculated with it missing. PREDVAR5 does not undertake this subtraction itself. However it can be easily carried out if the PREDVAR5 output file is imported into a spreadsheet.

4.14.1.2 Adding Observations

In this case the first observation type listed in the observation type list file must be a "base type", and must be named "base". As for the subtraction alternative, all observations cited in the observation type list file must also be cited in the PEST control file which defines the current inversion problem. The set of base observations are always employed in the notional inversion processes carried out by PREDVAR5. Other observation types are added to this base set in sequential PREDVAR5 predictive error variance calculation operations. The effect that each set of new observations has in reducing the error variance of the nominated prediction can then be judged by subtracting the error variance computed with inclusion of the new observation type in the notional inversion process from that computed on the basis of the base observation type alone. All observations contained within the PEST control file that do not belong to the base observation type, or to the observation type that is being currently processed, are assigned a weight of zero.

As for the observation subtraction option, prior information can be included in an observation type. However if, in the PEST control file read by PREDVAR5 which defines the current inversion problem, PEST is run in regularisation mode, a regularisation observation or prior information equation cannot be cited in the observation type listing file, as PREDVAR5 ignores these observations and prior information equations.

The base observation type can be empty if desired. In this case the contribution to error

variance made by the second term of the error variance equation (i.e. the "measurement noise term") is zero, and that made by the first term is the error variance under pre-calibration conditions (i.e. the pre-calibration predictive uncertainty). However other observation types cannot be empty in this case. Where the base observation type is empty, each listed observation type sequentially comprises the entirety of the tested calibration dataset.

The "observation addition" option is useful for assessing the relative worth of different data acquisition strategies. All possible additional observations are included in the PEST control file, along with the existing calibration dataset (the latter comprising the base observation type). PREDVAR5 then computes the predictive error variance with different observation types in turn added to the set of base observations. The user can then compare the relative efficacy of these observation types in lowering the error variance of the identified prediction. Note that a given observation can appear in more than one observation type.

4.14.2 Using PREDVAR5

PREDVAR5 commences execution with the prompts:-

```
Enter name of PEST control file:
Enter observation reference variance:
```

As for the other PREDVAR programs, the PEST control file provides specifications of the current inverse problem. It defines the parameters involved in that problem and their transformation status. It is assumed that a corresponding Jacobian matrix file (i.e. JCO file) exists; this is assumed to have the same filename base as the PEST control file, but to possess an extension of "*.jco*". It may have been created on a previous PEST run (possibly with the NOPTMAX variable set to -1 so that the run was carried out specifically for this purpose), or it may have been created using the JCO2JCO utility.

See the next section for a discussion of the reference variance.

PREDVAR5's next prompt us:-

```
Enter name of parameter uncertainty file:
```

This file provides the C(**p**) matrix; its format has been described elsewhere in this manual. Next PREDVAR5 prompts:-

```
Enter name of predictive sensitivity matrix file:
```

This is a file, in PEST matrix file format, containing the sensitivities of a prediction of interest to parameters cited in the PEST control file. Note the following.

1. Sensitivities to parameters additional to those existing in the PEST control file are ignored. Also, the ordering of parameters in the matrix file does not need to be the same as in the PEST control file.
2. If a parameter is log transformed or has other parameters tied to it, this must be reflected in the sensitivities supplied in this file.
3. If a parameter is cited in the PEST control file (but is not fixed or tied) but not in the parameter sensitivity vector, PREDVAR5 ceases execution with an appropriate error message.

Normally the predictive sensitivity vector will have been extracted from a JCO file using the JROW2VEC utility.

Next PREDVAR5 prompts:-

```
Enter name of singular value list file:
```

The role of this file is the same as its role in other PREDVAR programs. That is, it provides a list of singular values for which PREDVAR5 calculates error variance for each of the various observation types asked of it. It choses the minimum predictive error variance in each case. It is important to ensure that the range of singular values provided in this file is sufficient to encompass the predictive error variance minimum (a previous PREDVAR1 exercise may help in this regard). It is also a good idea for singular values to be sequential, and separated by unity. This minimises "granularity" in predictive error variance contributions when large numbers are subtracted from each other to form smaller ones.

PREDVAR5's next prompt is:-

```
Enter name of observation type list file:
```

The format of this file is described above. Then:-

```
Subtract list members or add to base observations [s/a]:
```

This is where the user selects between the two modes of PREDVAR5 operation described above. Its final prompt is:-

```
Enter name for predictive error variance output file:
```

The PREDVAR5 output file is similar to that of PREDVAR3 and PREDVAR4. It is normally best to import this file into a spreadsheet. For the observation addition option, subtract the predictive error variance calculated with each new set of nominated observations from the predictive error variance computed with no observations subtracted (the latter can be computed using an empty observation type) to calculate the effect that each observation type would have in reducing predictive error variance from its current value. For the observation subtraction option, subtract the predictive error variance computed with no observations subtracted from that calculated with each nominated observation type subtracted to compute the effect that each observation type has in achieving the predictive error variance of the currently calibrated model.

4.14.3 Observation Weights

Unlike other predictive error variance analysis programs supplied with the PEST suite, PREDVAR5 does not prompt for an observation uncertainty file. Instead PREDVAR5 (and all of the other PREDVAR programs) assumes that observation weights supplied in the PEST control file are proportional to the inverse of measurement standard deviations. Alternatively, a covariance matrix may be supplied for one or a number of observation groups cited in the PEST control file (using normal PEST control file protocol). A user's original idea of measurement variance may have been altered through a previous calibration exercise, from which a non-unity reference variance was obtained. This reference variance can be supplied to PREDVAR5 in response to the second of its prompts. If a value other than unity is supplied, PREDVAR5 effectively multiplies all covariance matrices cited within the PEST control file by this reference variance; all weights are effectively divided by the square root of this reference variance.

4.14.4 Parameter Scaling

Parameters should be scaled using the SCALEPAR utility before using any of the PREDVAR programs. This ensures that there are no rises in predictive error variance before the latter starts to fall as singular values are increased (or that these rises are small, as can happen where some parameters are correlated with each other through non-diagonal terms of the

C(**p**) matrix). It also promotes minimization of predictive error variance. You don't actually need to run PEST using the PEST calibration dataset generated by SCALEPAR to obtain scaled parameters; however you do need to allow SCALEPAR to compute the scaled parameter Jacobian matrix. Furthermore you need to adjust parameter standard deviations such that they are all unity; similarly, parameter covariance matrices should be adjusted so that their diagonal elements are all unity in harmony with SCALEPAR's re-scaling calculations; off-diagonal elements must be adjusted accordingly. Predictive sensitivities must be scaled in the same way that observation sensitivities are scaled.

4.15 PREDVAR1A

The functionality of PREDVAR1A is very similar to that of PREDVAR1. However there are a number of noteable differences, these being as follows.

1. PREDVAR1A does not undertake singular value decomposition of the $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ matrix. Rather, it undertakes singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{X}$. Where there are fewer observations than parameters, this is numerically far swifter (especially where there are many parameters).
2. It does not read or write an SVD file. (It is assumed that use of PREDVAR1A will take place where parameter numbers are large; hence the SVD file would be very large indeed.)
3. PREDVAR1A calculates error variance terms for not just one, but for many predictions.

Prompts, and typical responses, to PREDVAR1A are displayed below.

```
Enter name of PEST control file: case.pst
Enter observation reference variance: 1.0

Enter name of parameter uncertainty file: param.unc
Enter name of predictive sensitivity matrix list file: predfile.lst

Enter no. of singular values before truncation [<Enter> if no more]: 0
Enter no. of singular values before truncation [<Enter> if no more]: 1
Enter no. of singular values before truncation [<Enter> if no more]: 2
Enter no. of singular values before truncation [<Enter> if no more]:
```

Part of a "predictive sensitivity matrix list file" is shown below.

```
ar10.vec  ar10.sen
ar9.vec   ar9.sen
ar8.vec   ar8.sen
ar7.vec   ar7.sen
```

A predictive sensitivity matrix list file.

Each line of a predictive sensitivity matrix list file must contain two entries. The first is the name of a predictive sensitivity file. The second is the name of the file to which predictive variances for all nominated singular values are to be written for the prediction whose parameter sensitivities are contained in the first file of the line. The following should be noted.

1. Blank lines are permissible in a predictive sensitivity matrix list file.
2. If a filename contains a blank character, its name should be enclosed by quotes.

4.16 PCOV2MAT

4.16.1 General

The purpose of PCOV2MAT is to facilitate the calculation of model predictive error variance in over-determined inversion contexts; these are contexts where regularisation is not employed as part of the inversion process. In these circumstances parameter error variance/covariance is completely described by the parameter covariance matrix produced as an outcome of the parameter estimation process. This is because, under these circumstances, the resolution matrix **R** of equation 4.1 is in fact the identity matrix **I**. If the measurement weight matrix **Q** is proportional to the inverse of the measurement noise matrix C(**ε**), then at the end of the parameter estimation process this can be updated through multiplication by the reference variance σ^2_r calculated from the minimized objective function Φ_{min} as:-

$$\sigma_r^2 = \Phi_{min}/(n\text{-}m) \qquad (4.13)$$

where *n* and *m* are the number of observations and parameters respectively featured in the parameter estimation process. Thus **Q** now becomes:

$$\mathbf{Q} = \sigma_r^2 C^{-1}(\boldsymbol{\varepsilon}) \qquad (4.14)$$

and the parameter error covariance matrix (or simply "parameter covariance matrix" under these over-determined circumstances) can be calculated as:-

$$C(\mathbf{p}\text{-}\underline{\mathbf{p}}) = \sigma_r^2(\mathbf{X}^t\mathbf{Q}\mathbf{X})^{-1} \qquad (4.15)$$

The parameter error covariance matrix is recorded by PEST at the end of its run record file *case.rec*; it is also calculated intermittently by PEST and recorded in the matrix file *case.mtt*. In both of these cases *case* is the filename base of the PEST control file on which the current parameter estimation exercise is based.

If a prediction has parameter sensitivities **y**, then the error variance of that prediction can be calculated as:-

$$\sigma^2_{s\text{-}\underline{s}} = \mathbf{y}^t C(\mathbf{p}\text{-}\underline{\mathbf{p}})\mathbf{y} \qquad (4.16)$$

4.16.2 Using PCOV2MAT

PCOVMAT is run using the command:-

```
pcovmat pestfile matfile
```

where:-

pestfile is the name of a PEST run record file or matrix file containing a covariance matrix, and

matfile is the name of the matrix file to which the matrix will be written; see Section 4.4.3 for a description of the format of this file.

The following should be noted.

1. Only adjustable parameters are cited in a parameter covariance matrix; fixed or tied parameters are not cited.
2. It is preferable to use the parameter covariance matrix from a run record file over that recorded in a matrix file as the former is calculated using sensitivities pertaining to optimized parameters, whereas the covariance matrix recorded in the latter file is calculated on the basis of sensitivities pertaining to the latest PEST iteration.

3. A covariance matrix is not recorded in the matrix file unless the ICOV variable in the "control data" section of the PEST control file is set to 1.

Once the parameter error covariance matrix has been translated by PCOV2MAT to matrix file format, the error variance of a particular prediction can be calculated on the basis of equation 4.16 using the MATQUAD utility documented in the next section. Prediction sensitivities can be extracted from a Jacobian matrix file using the JROW2VEC or JROW2MAT utilities; in the latter case the MATTRANS utility must be used before MATQUAD is employed to calculate predictive error variance.

4.17 SCALEPAR

4.17.1 General

SCALEPAR was written primarily to assist in the use of programs such as PREDVAR1 to PREDVAR5. However there is no reason why it cannot also be employed in normal parameter estimation. The purpose of SCALEPAR is to re-formulate an inversion problem in terms of scaled parameters rather than native parameters, with parameters scaled according to their standard deviations. This can result in smaller predictive error variance when undertaking regularised inversion, and may forestall rises in predictive error variance incurred by the calibration process which can sometimes occur where the "innate variability" of parameters (as expressed by the C(**p**) matrix of equation 4.1) is such that individual parameters are highly correlated with each other and have very different variances. Where a parameter is log-transformed in the inversion process, and where pertinent elements of C(**p**) thus refer to the log (to base 10) of that parameter, SCALEPAR takes this into account.

In re-formulating the inverse problem, SCALEPAR writes a new PEST control file in which scaled parameters have an initial value of zero and are permitted to vary between -3.0 and 3.0. Because parameters are scaled by their standard deviations, they are thus allowed to vary by three standard deviations either side of their most likely value of zero, this corresponding to a native parameter value also equal to its most likely value. It is assumed that the most likely value of each native parameter corresponds to its initial value as supplied in the original PEST control file. Thus, when writing this PEST control file, the user must ensure that initial parameter values are indeed "most likely" parameter values, that is "expected parameter values" in the statistical sense.

If a parameter is log-transformed in the original PEST control file, scaled parameters defined in the SCALEPAR-generated PEST control file are actually the scaled logs of such parameters. Once again, a scaled parameter value of zero corresponds to a native untransformed parameter value equal to that of the initial value supplied in the original PEST control file.

Whether parameters are log-transformed or not in the original parameter estimation problem, back transformation from scaled parameter values (as seen by PEST) to native parameter values (as seen by the model) is undertaken by the PAR2PAR utility documented in the PEST manual, which is run as part of the model ahead of any other components of the model. The new PEST control file written by SCALEPAR cites only one template file, this corresponding to a PAR2PAR input file. After it has "de-scaled" parameter values PAR2PAR writes native parameter values to all model input files which require them according to specifications set out in the original PEST control file.

SCALEPAR also provides the option of writing a Jacobian matrix corresponding to the new

PEST control file. Sensitivities which occupy the elements of this new Jacobian matrix are computed from sensitivities occupying corresponding elements of a Jacobian matrix file corresponding to the original PEST control file (if such a Jacobian matrix exists).

Another option offered by SCALEPAR is the writing of a new parameter uncertainty file, this pertaining to scaled parameters. By definition, the standard deviation of each scaled parameter is one.

4.17.2 Calculating the Resolution Matrix of Native Parameters

Suppose that unscaled parameters cited in the original PEST control file are designated by the vector **p**, and that their scaled counterparts are designated by the vector **q**. Thus:-

$$\mathbf{q} = \mathbf{S}\mathbf{p} \qquad (4.17)$$

where **S** is a diagonal "scaling matrix" whose elements are the inverse of the standard deviations of the parameters to which they pertain. Note that offsets (i.e. parameter initial values) are ignored in this equation for the sake of simplicity; however they are included in the actual SCALEPAR transformation process.

Suppose that explicit or notional (for example using PREDVAR1) regularised inversion has been carried out, and that a resolution matrix **R** has been calculated linking estimated scaled parameters $\underline{\mathbf{q}}$ to their real-world (but unknown) counterparts **q**. Thus:-

$$\underline{\mathbf{q}} = \mathbf{R}\mathbf{q} \qquad (4.18)$$

Using (4.17), together with the relationship:-

$$\mathbf{p} = \mathbf{S}^{-1}\mathbf{q}$$

derived from (4.17), it is possible to compute the relationship between estimated native model parameters and their real-world counterparts as:-

$$\underline{\mathbf{p}} = \mathbf{S}^{-1}\underline{\mathbf{q}} = \mathbf{S}^{-1}\mathbf{R}\mathbf{q} = \mathbf{S}^{-1}\mathbf{R}\mathbf{S}\mathbf{p} \qquad (4.19)$$

Thus the resolution matrix $\mathbf{R}'$ linking $\underline{\mathbf{p}}$ to **p** is:-

$$\mathbf{R}' = \mathbf{S}^{-1}\mathbf{R}\mathbf{S} \qquad (4.20)$$

SCALEPAR records the **S** and $\mathbf{S}^{-1}$ matrices in matrix file format discussed in the following section. The utility program MATPROD can then be employed to implement the matrix multiplication of (4.20) if necessary.

4.17.3 Running SCALEPAR

SCALEPAR is run by typing its name at the command prompt. No command line arguments are required because SCALEPAR asks the user specifically for each item of information that it requires.

SCALEPAR begins execution with the prompt:-

```
Enter name of existing PEST control file:
```

Supply the name of an existing PEST control file in response to this prompt. SCALEPAR requires that this file meet the following specifications.

1. No prior information must be cited in this file.
2. The SCALE associated with each parameter (in the "parameter data" section of the existing PEST control file) must be 1.0, while each parameter OFFSET must be 0.0.

3. The model command line must cite a batch or script file. In the PC environment this command must have an extension of "*.bat*".
4. Only a single command must be employed to run the model for finite-difference calculation of derivatives with respect to all adjustable parameters; thus PEST's multiple command line functionality must not be invoked, and the control variable NUMCOM must be set to 1, or omitted.
5. Derivatives must be calculated by finite differences and not supplied by the model through an external file. Thus the JACFILE control variable must be set to zero or omitted.

If any of these conditions are violated, SCALEPAR will cease execution with an appropriate error message.

SCALEPAR's next prompt is:-

```
Enter name of parameter uncertainty file:
```

The format of this file is described in the documentation to program PARAMERR above. Uncertainty data (i.e. data comprising the C(**p**) matrix) must be supplied in this file for each adjustable (i.e. non-fixed and non-tied) parameter cited in the original PEST control file. As usual, if a parameter is log-transformed, pertinent elements of C(**p**) must pertain to the log of that parameter.

SCALEPAR next asks:-

```
Enter name for new PEST control file:
```

in response to which the name of the PEST control file which SCALEPAR must write should be provided.

Once it has been provided with this name, SCALEPAR issues a series of prompts, to which the user can respond simply by pressing the <Enter> key in each case to accept the SCALEPAR default. The prompts are:-

```
Enter name for PAR2PAR input file (<Enter> if p2p###.dat):
Enter name for template of this file (<Enter> if p2p###.tpl):
Enter name for parameter value file (<Enter> if p###.par):
Enter name for template of this file (<Enter> if p###.tpl):
Enter name for scaling matrix file (<Enter> if scale.mat):
Enter name for inverse scaling matrix file (<Enter> if iscale.mat):
Run model in silent or verbose mode  [s/v] (<Enter for "s"):
```

As discussed above, parameter "de-scaling" is actually undertaken by PAR2PAR. Prior to each model run the modified PEST control file informs PEST that it must write a PAR2PAR input file in which scaled parameter values are listed. PAR2PAR then computes native parameter values from these scaled values (and performs inverse log transformation if necessary) and writes them to appropriate model input files. SCALEPAR ensures that the PAR2PAR input file that PEST writes contains the names of all model template and corresponding input files involved in the inversion process as listed in the original PEST control file. In fact SCALEPAR writes a template file of the PAR2PAR input file for the use of PEST, and informs PEST of the name of the PAR2PAR input file to which this template file corresponds. The names of the PAR2PAR template file and corresponding PAR2PAR input file can be supplied in response to the first two of the above set of prompts.

As is documented in the PEST manual, as PEST carries out the parameter estimation process, it records optimised parameter values in a "parameter value file". If PEST is run on the basis

of the SCALEPAR-generated PEST control file, optimised *scaled* parameter values are indeed recorded in this file. This is of limited use to the user, however, who is normally more interested in optimised *native* parameter values than scaled parameter values. Unfortunately, native parameter values are known only to PAR2PAR, and not to PEST. To overcome this difficulty, PAR2PAR is instructed to write, on each occasion that it runs, a file that has the same format as a PEST parameter value file (and can thus be used by utility programs such as PARREP), containing native parameter values employed on that model run. The name of this file can be supplied by the user in response to the third of the above prompts. SCALEPAR provides PAR2PAR with a template file through which it can write this file; the name of this template file can be supplied in response to the fourth of the above prompts.

It is important to note that, unlike the parameter value file recorded by PEST, the parameter value file recorded by PAR2PAR does not contain optimised parameter values up to the stage of the parameter estimation process at which it is viewed. Because PAR2PAR writes model input files on the basis of template files on each occasion that the model is run, the PAR2PAR-generated parameter value file contains native parameter values employed only on the last model run. However if PEST has completed the parameter estimation process (or is halted using the "stop with statistics" option), it will have completed a final model run employing optimised parameter values. Thus at the end of the parameter estimation process the PAR2PAR-generated parameter value file will, in fact, contain optimised native parameter values.

As stated above, SCALEPAR generates matrix files containing the $\mathbf{S}$ and $\mathbf{S}^{-1}$ scaling and inverse scaling matrices. The names of these matrix files can be supplied in response to the fifth and sixth of the above prompts.

The model as run by the modified PEST control file must itself be modified so that PAR2PAR can be run ahead of the actual model in order to generate native parameters from the scaled parameters employed by PEST, and write these to model input files. SCALEPAR adds the command to run PAR2PAR to the batch or script file which was originally employed for running the model. The name of this modified batch file is always *scalebatch.bat*. If the "silent" option is selected in response to the last of the above prompts, PAR2PAR will direct its screen output to the "nul" file instead of to the screen, thus preventing PEST's screen output from being scrolled away out of sight.

SCALEPAR's next prompt is:-

```
Write a JCO file for this case [y/n] (<Enter> for "n"):
```

If the response to this prompt is "y", and if a Jacobian matrix file complementary to the original PEST control file exists, SCALEPAR will write a Jacobian matrix file which complements the new PEST control file. This can then be used in conjunction with members of the PREDVAR suite in examining predictive variances. Alternatively, it can be used in the first iteration of a PEST run undertaken on the basis of the new PEST control file if that run is initiated with the "/i" switch. (In the latter case it will eventually be overwritten by PEST as it optimises parameters on the basis of the new PEST control file.)

Next SCALEPAR asks:

```
Write scaled uncertainty file? [y/n]: (<Enter> for "n"):
```

If the response to this question is "y", SCALEPAR promots for a suitable name for this file:

```
Enter name for scaled uncertainty file: (<Enter> if p###.unc):
```

If the original uncertainty file cites no covariance matrices, this is all that SCALEPAR needs

to know. Uncertainties are recorded as a list of parameter standard deviations, all of which are 1.0. Howevr if at least one covariance matrix was cited in the original parameter uncertainty file, SCALEPAR asks:

```
Enter name for cov mat file cited therein (<Enter> if p###.cov):
```

In the this case, all parameter uncertainty is recorded as a single covariance matrix, the diagonal elements of which are 1, the off-diagonal elements being zero, or scaled in accordance with the original covariance submatrices from which they were read.

4.17.4 The New PEST Control File

Certain features of the control file generated by SCALEPAR are worthy of mention.

Scaled parameters cited in the SCALEPAR-generated PEST control file are given the same names as their unscaled counterparts in the original PEST control file. Only adjustable parameters are cited in this file. Tied and fixed parameters are still involved in the modified inversion process; however the values of fixed parameters, and the multipliers through which tied parameters are linked to their parent parameters, are recorded in the SCALEPAR-generated PAR2PAR template file. Hence PAR2PAR, and not PEST, accommodates the assignment of values for these parameters to the model.

It was mentioned above that scaled parameters are provided with an initial value of zero, and upper and lower bounds of -3.0 and 3.0. A value of zero for a scaled parameter corresponds to a native parameter value equal to its initial value as supplied in the original PEST control file; the parameter offsetting required to achieve this is taken care of by PAR2PAR through the pertinent equations written by SCALEPAR to the PAR2PAR template file. An inspection of the SCALEPAR-generated PEST control file will reveal, however, that scaled parameters themselves are actually offset by 10.0 from zero, and provided with lower bounds of -7.0 and upper bounds of 13.0. This circumvents problems that are sometimes encountered with the imposition of relative change limits on parameter upgrades through use of the RELPARMAX control variable for parameters that are close to zero. In short, relative-limited parameters can be upgraded towards zero very rapidly; however they can only move back from zero relatively slowly because any change in the value of a near-zero parameter is large relative to its current value. This can slow the parameter estimation process considerably in some circumstances.

All scaled parameters in the SCALEPAR-generated PEST control file are declared as relative-limited; RELPARMAX is provided with a value of 0.1 (this being relative to their offset values of 10.0). This, and parameter OFFSET values, can be altered if desired by direct editing of the SCALEPAR-generated PEST control file.

All parameters are assigned to a single parameter group in the SCALEPAR-generated PEST control file. This group is assigned an absolute increment for the purpose of derivatives calculation, this increment being 0.01. This, and any other aspect of the PEST control file generated by SCALEPAR, can also be altered by direct editing of this file if desired.

4.18 REGERR

4.18.1 General

REGERR evaluates the covariance matrix of regularisation-induced model output error. For present purposes this is defined as:-

$$\boldsymbol{\tau} = \mathbf{X}(\mathbf{I} - \mathbf{R})\mathbf{p} \quad (4.21)$$

so that:-

$$C(\boldsymbol{\tau}) = \mathbf{X}(\mathbf{I} - \mathbf{R})C(\mathbf{p})(\mathbf{I} - \mathbf{R})^t\mathbf{X}^t \quad (4.22)$$

In this equation C(**p**) is the pre-calibration parameter covariance matrix, **R** is the resolution matrix for the current inversion problem, and **X** is the Jacobian matrix. REGERR obtains **R** from a RESPROC output file and **X** from a Jacobian matrix file (i.e. a JCO file). Presumably these will have been produced on the basis of the same PEST input dataset. However REGERR only tests that the number of parameters cited in the RESPROC output file and the number of parameters cited in the Jacobian matrix file are the same. The number of observations can differ between these two files. Thus the Jacobian matrix file can pertain to model outputs that differ from those employed in the calibration process if desired; the covariance matrix of regularisation-induced predictive error can thereby be calculated. The latter is very similar to the covariance matrix of predictive error; however it lacks the contribution to this error from measurement noise.

4.18.2 Running REGERR

REGERR is run simply by typing its name at the screen prompt; it then prompts the user specifically for its input data requirements. Prompts, and typical replies, are illustrated below:-

```
Enter name of RESPROC output file: case.rpo

Enter name of parameter uncertainty file: param.unc

Enter name of Jacobian matrix file: case.jco

Enter name for output covariance matrix file: cov.mat

- reading RESPROC output file case.rpo...
- file case.rpo read ok.

- reading Jacobian matrix file case.jco...
- Jacobian matrix file case.jco read ok.

- reading parameter uncertainty data...
- parameter uncertainty data read ok.

- calculating regularisation-induced output error covariance matrix...
- file cov.mat written ok.
```

The format of a parameter uncertainty file is discussed earlier in this document. The covariance matrix file written by REGERR employs the matrix file protocol discussed in the next chapter of this document.

4.19 REGPRED

4.19.1 General

The purpose of REGPRED is to write a set PEST input dataset in which "regularised nonlinear predictive uncertainty analysis" is carried out. This term is employed to embody the

nonlinear extension of the theory presented in Section 4.1.

Use of REGPRED is based on the premise that model calibration has been accomplished through regularised inversion. Notionally, the regularised inversion process can be conceived of as subdividing parameter space into two distinct subspaces – the "calibration solution space" and the "calibration null space". Parameter combinations within the former subspace are informed by the calibration process; those within the latter subspace are not. The potential errors associated with parameter combinations associated with the former subspace are a function of measurement noise C(**ε**). The potential errors associated with the latter parameter combinations are a function of the "innate variability" (possibly conditioned by measurements or expert knowledge) of real-world parameters (i.e. system hydraulic properties) as encapsulated in the C(**p**) matrix discussed extensively throughout this addendum. For an uncalibrated model, it is this variability alone which determines parameter and predictive uncertainty.

REGPRED writes a PEST input dataset in which PEST is asked to run in "predictive analysis" mode. In doing so it maximises or minimises a key model prediction while maintaining the objective function at or below a user-specified value. In the REGPRED-generated PEST control file, the objective function is defined in such a way that it encompasses both of the constraints discussed above. That is, it constrains parameter combinations lying within the calibration null space to respect the fact that they must be realistic (at a certain probability level), at the same time as it ensures that the model does not "become uncalibrated" by employing parameter sets that result in a mismatch between model outputs and field measurements that is not justified by the noise content of those measurements.

4.19.2 Theory

4.19.2.1 Calibration Null Space Constraints

Where regularisation is achieved through truncated singular value decomposition (TSVD), and if the Marquardt lambda is zero, then the neat subdivision of parameter space into estimable and non-estimable subspaces is exactly achieved. This is also accomplished if parameter estimation is achieved through SVD-assist and if no Tikhonov regularisation is employed in the SVD-assisted parameter estimation process. If regularisation is achieved by other means, then the subdivision of parameter space into orthogonal estimable and non-estimable subspaces is not exact but approximate. Use of REGPRED, however, assumes that the approximation is good enough to allow formulation of parameter and model output constraints enforced during the predictive analysis process, in terms of these subspace concepts.

Moore and Doherty (2005) show that parameter error of a calibrated model (i.e. the discrepancy between true model parameters **p** and estimated model parameters $\underline{\mathbf{p}}$) is described by the equation:-

$$\mathbf{p}-\underline{\mathbf{p}} = (\mathbf{I} - \mathbf{R})\mathbf{p} - \mathbf{G}\boldsymbol{\varepsilon} \qquad (4.23)$$

where **R** and **G** are the resolution and "G" matrices calculated as a by-product of the regularised inversion process.

Where regularised inversion is achieved through TSVD then:-

$$\mathbf{R} = \mathbf{V}_1\mathbf{V}^t_1$$

$$\mathbf{I}\text{-}\mathbf{R} = \mathbf{V}_2\mathbf{V}_2^t$$

and

$$\mathbf{G} = \mathbf{V}_1 \mathbf{E}^{-1}{}_1\mathbf{V}^t{}_1\mathbf{X}^t\mathbf{Q}$$

where the columns of $\mathbf{V}_1$ and $\mathbf{V}_2$ are the orthonormal eigenvectors of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ spanning the calibration solution and null spaces respectively as discussed extensively in this addendum. From (4.23) the covariance matrix of model parameter error C($\mathbf{p}$-$\underline{\mathbf{p}}$) can be calculated as:-

$$C(\mathbf{p}\text{-}\underline{\mathbf{p}}) = (\mathbf{I}–\mathbf{R})C(\mathbf{p})(\mathbf{I}\text{-}\mathbf{R})^t + \mathbf{G}C(\boldsymbol{\varepsilon})\mathbf{G}^t \qquad (4.24)$$

To explore the error range of a prediction, the constrained maximisation/minimisation process to be described shortly focuses on $\mathbf{p}$-$\underline{\mathbf{p}}$ rather than $\mathbf{p}$ in the imposition of constraints on parameters as these parameters are varied in order to maximise or minimise a prediction. Moreover, constraints on parameters will only be applied in the calibration null space, for constraints applied to model outputs will effectively constrain parameter combinations that lie within the calibration solution space.

It is assumed that a set of calibrated parameters $\underline{\mathbf{p}}$ already exists. The extent to which deviations from this set are tolerable at a certain probability level is defined by C($\mathbf{p}$-$\underline{\mathbf{p}}$) of equation 4.24. Where such deviations are along directions spanned by the null space, tolerance of movement of $\mathbf{p}$ (and hence of $\mathbf{p}$-$\underline{\mathbf{p}}$) is governed by the first term of equation 4.24; the governing probability distribution in this case is C($\mathbf{p}$). To the extent to which changes in parameter values are incurred along directions spanned by the solution space, it is the second term of 4.24 which exerts these constraints; for this term the governing probability distribution is C($\boldsymbol{\varepsilon}$), that is, the stochastic description of measurement noise.

REGPRED asks the user to nominate the number of dimensions comprising the solution space to the current parameter estimation problem. Where parameter estimation is implemented using SVD (without the Marquardt lambda), this will be equal to the number of singular values at which truncation occurs. For other types or regularisation the user can employ the MATSVD utility to undertake singular value decomposition of the $\mathbf{R}$ matrix for the current problem (available through use of the RESPROC and RESWRIT utilities). Inspection of the eigenvalue matrix forthcoming from this decomposition will reveal the number of leading eigenvalues of $\mathbf{R}$ that are seriously nonzero. This number then defines the dimensionality of the solution space. (Ideally, eigenvalues will be either one or zero. But to the extent that the Marquardt lambda or Tikhonov regularisation impacts upon the parameter estimation process, eigenvalues will differ from these ideals.)

REGPRED itself undertakes singular value decomposition of the resolution matrix $\mathbf{R}$. In doing so, it calculates $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$ such that:-

$$\mathbf{R}=\mathbf{U}\mathbf{S}\mathbf{V}^t \qquad (4.25)$$

For the current set of parameter values $\mathbf{p}$ employed in the predictive analysis maximisation/minimisation process it computes:-

$$\mathbf{g} = \mathbf{U}_2^t(\mathbf{p}\text{-}\underline{\mathbf{p}}) \qquad (4.26)$$

where $\mathbf{U}_2$ is comprised of the last m-k columns of $\mathbf{U}$, k being the number of dimensions in the solution space as provided by the user and m being the total number of parameters employed in the parameter estimation exercise. $\mathbf{U}_2$ approximately spans the calibration null space (exactly so for TSVD parameter estimation). The $\mathbf{g}$ matrix has m-k elements,

From (4.23):-

$$\mathbf{g} = \mathbf{U}_2^t (\mathbf{p}-\underline{\mathbf{p}}) = \mathbf{U}_2^t (\mathbf{I} - \mathbf{R})\mathbf{p} - \mathbf{U}_2^t \mathbf{G}\boldsymbol{\varepsilon} \quad (4.27)$$

The second term on the right side of 4.27 is assumed to be zero; it is in fact exactly zero when regularisation in undertaken using TSVD. Hence the covariance matrix of **g** is given by:-

$$C(\mathbf{g}) = \mathbf{U}_2^t C(\mathbf{p}-\underline{\mathbf{p}})\mathbf{U}_2 = \mathbf{U}_2^t (\mathbf{I} - \mathbf{R})C(\mathbf{p})(\mathbf{I} - \mathbf{R})^t\mathbf{U}_2 \quad (4.28)$$

In the predictive analysis PEST control file written by REGPRED, all elements of the **g** matrix are read as observations (the calculation of **g** through equation 4.26 is implemented through a modification of the model to be described below). They are assigned an "observed value" of zero and collectively assigned to an observation group of their own. This group is then assigned the covariance matrix C(**g**) which is computed by REGPRED using equation 4.28 and written to a covariance matrix file ready for the use of PEST. Because the elements of **g** thus comprise part of the observation dataset, deviations of **g** from zero as computed on the basis of a given parameter set contribute to the objective function. Thus in constraining the objective function during the maximisation/minimisation process implemented by PEST's predictive analyser, parameter movement within the calibration null space is also constrained.

As an alternative to the above procedure, instead of reading a resolution matrix, REGPRED prompts the user for the contents of a singular value decomposition file. Such a file (written by PEST when singular value decomposition is activated) contains the **V** matrix (i.e. the matrix of eigenvectors of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$**),** provided the EIGWRITE variable in the "singular value decomposition" section of the pertinent PEST control file was is set to 1. REGPRED extracts $\mathbf{V}_2$ from this matrix (i.e. the set of eigenvectors spanning the calibration null space) and uses $\mathbf{V}_2^t$ in place of $\mathbf{U}_2^t$ and $\mathbf{U}_2^t$ **(I – R)** in the above procedure. Where regularised inversion was implemented using TSVD, and thus (**I - R**) is equal to $\mathbf{V}_2\mathbf{V}_2^t$, all of these are equivalent. Where regularised inversion was implemented using SVD-assist, with super parameters defined on the basis of decomposition of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ (which is always the case unless external super parameters are defined), this represents a fast way of obtaining the projections of model parameters onto the null subspace and of thus isolating the contribution of the first term of 4.23 to parameter error. This is because, for SVD-assist, $\mathbf{V}_2^t\mathbf{G}$ is equal to zero (see equation 4.8b) and thus the second term of 4.23 becomes zero when pre-multiplied by $\mathbf{V}_2^t$.

4.19.2.2 Calibration Solution Space Constraints

Error in the estimation of those combinations of parameters that lie within the calibration solution space arises solely from measurement noise. Thus a stochastic description of that error rests ultimately on a stochastic description of measurement noise, i.e. C(**ε**).

Suppose that the estimated parameter set $\underline{\mathbf{p}}$ when supplied to the model gives rise to model outputs $\underline{\mathbf{o}}$. As parameters are altered through the prediction maximisation/minimisation process a set of parameters $\underline{\mathbf{p}}+\delta\underline{\mathbf{p}}$ may be calculated. These may, in turn, result in model outputs $\underline{\mathbf{o}} + \delta\underline{\mathbf{o}}$. Now if $\delta\underline{\mathbf{o}}$ is "unlikely" at a certain significance level in terms of measurement noise, then $\underline{\mathbf{o}} + \delta\underline{\mathbf{o}}$ will be statistically different from $\underline{\mathbf{o}}$ at this same significance level. Thus also at this same significance level, the alterations $\delta\underline{\mathbf{p}}$ to the calibrated parameter set $\underline{\mathbf{p}}$ will be statistically unlikely. As a consequence it is then unlikely that $\underline{\mathbf{p}}+\delta\underline{\mathbf{p}}$ can be considered as an alternative to $\underline{\mathbf{p}}$ at the same significance level, and so $\delta\underline{\mathbf{p}}$ must be constrained to prevent this.

This can be viewed in another way. Suppose that measurement noise for the current calibration dataset is a stochastic realisation based on the covariance function C(**ε**). Let this realisation be denoted as $\boldsymbol{\varepsilon}_1$. Suppose also that "true" system behaviour is encapsulated in the vector **o** so that the measurement dataset **h** is expressed as $\mathbf{o}+\boldsymbol{\varepsilon}_1$. We have assumed that a

model can perfectly replicate true system behaviour if provided with a correct parameter set. Suppose that this parameter set (possibly after projection onto the calibration solution space) is **p**. Now if any parameter set **p**+δ**p** gives rise to a set of model outputs **o** + δ**o** to which the same measurement noise is added, the resulting calibration dataset will be **h** + δ**o**. If δ**o** is a statistically unlikely random variable as assessed using the covariance matrix C(**ε**), then **h** + δ**o** is significantly different from **h** and the parameter difference δ**p** is thus discernable from the calibration dataset, notwithstanding the noise content of this data. **p**+δ**p** can thus *not* be considered as a parameter set which "calibrates" the model at a certain significance level because it results in a discernable difference in model outputs, at that same significance level.

Following this logic, when implementing maximization/minimization of a user-specified prediction, constraints are not implemented directly on model-to-measurement misfit. Rather they are imposed on the change in model outputs induced through the maximization/minimization process. If this change is considered unlikely at a certain significance level as assessed in terms of the covariance matrix C(**ε**) of measurement noise, then so too are the parameters required to achieve this, at the same significance level.

4.19.2.3 Imposition of all Constraints

With the two sets of constraints now identified, the manner of their imposition can be defined.

As will be discussed below, REGPRED assumes that the user has built a PEST control file in which initial parameter values are optimised parameter values, and in which initial observation values are model outputs calculated using these optimised parameter values. Any regularisation information present within this file is ignored. However the presence of regularisation in the preceding parameter estimation process is "felt" through the values of the optimised parameters, and through use of the corresponding resolution matrix in defining **g** of equation 4.27. It is assumed that this resolution matrix has already been calculated by the user.

Observations within the existing PEST control file that are not regularisation observations are transferred to the new PEST control file that is written by REGPRED. However, as discussed, REGPRED assumes that their values are now perfectly matched to the initial parameter values supplied in this file (this can be achieved using the OBSREP utility); thus all residuals corresponding to all observations are initially zero. REGPRED assigns weights to these observations which are the inverse of the uncertainties of the associated measurements, measurement uncertainty being read from an "observation uncertainty file". (It should be noted here that observation weights as employed in the predictive analysis process can thus be different from those employed in the parameter estimation process. The benefit of using weights that are "tuned" to a particular parameter estimation problem has been discussed in Moore and Doherty, 2005). Note that REGPREG, as presently programmed, does not allow a covariance matrix to be used to characterise the uncertainty of field measurements.

The observation dataset written to the new PEST control file is supplemented by the elements of **g** as defined in 4.27; the "observed value" or each element of **g** is zero. These new observations are assigned to an observation group of their own, for which the C(**g**) covariance matrix of 4.28 is employed for specification of uncertainty. As defined above **g** is nonzero only where parameters depart from their initial values in combinations that lie within the calibration null space.

An objective function is thus formed comprised of modified measurements and modified parameter differences projected onto the calibration null space. The objective function is zero at the commencement of the predictive analysis process and will only become non-zero as parameters depart from their previously estimated values. If such departures take place in combinations that lie within the calibration solution space, model outputs will be affected. If they take place in combinations that lie within the calibration null space, the elements of **g** will be affected. All observations are assigned weights (or a covariance matrix in the case of **g**) which are in exact accord with their statistical distributions. If the noise associated with measurements is Gaussian, and if C(**p**) is multiGaussian, the square root of the objective function thus measures, in normalised observation space, the distance of departure of the collective model output and **g** dataset from its optimum value of **0** (for which the objective function is zero). The square root of the objective function is thus a normalized normal variate and can be employed as a means of setting a significance level on these departures. Thus, for example, if the objective function is raised to 9 for a certain parameter set, then the distance, in normalised observation space, of the collective set of "observations" (including the **g** component of these observations) from their optimum value of **0** is 3 standard deviations. Thus at this point the "observations" are statistically different from their optimised counterparts at the 99.7% confidence level. Given the definition of these observations, this means that there is a very low likelihood that the current parameter set satisfies the stochastic requirements of C(**p**) and that model output deviations calculated on the basis of the current parameter set satisfy the stochastic requirements of C(**ε**); thus any prediction made on the basis of that parameter set is equally unlikely.

4.19.3 Preparations for a REGPRED Run

4.19.3.1 Obtaining the Resolution Matrix

Prior to running REGPRED a regularised inversion exercise must have been carried out. Regularisation could have been achieved using pure TSVD (in which case PEST was run in "estimation" mode), using Tikhonov constraints, and/or using SVD-assist. It is assumed that the RESPROC utility has been run after this, and that RESWRIT has been run to produce a resolution matrix, stored in file of its own using the matrix file format discussed in Section 5 of this addendum. Note that if regularised inversion is carried out using SVD-assist, a second set of parameter sensitivities may have been calculated based on optimised parameters in order to provide more accuracy in computation of the resolution matrix. See documentation of RESPROC for more details.

4.19.3.2 Building an "Optimised PEST Control File"

The next task, prior to running REGPRED, is to build a PEST control file in which initial parameter values are optimal, and in which "observations" are in fact the set of model outputs corresponding to these optimised parameters. The former can be achieved using the PARREP utility (and may have been done already to calculate sensitivities on the basis of optimised parameters after an SVD-assist run as discussed above). The second of the above tasks can be carried out using OBSREP.

After having run both of these programs an "optimised PEST control file" will have been produced. If NOPTMAX is set to 0 in this file and PEST is run in order to carry out just one model run, that model run should show a measurement objective function of zero.

There is no need to remove regularisation observations and/or regularisation prior information from this file, for REGPRED will take care of this. However the following

should be noted.

1. If the optimised PEST control file does not instruct PEST to run in "regularisation" mode (as inherited from the files from which it was built), then the name of no observation group should begin with the string "regul".
2. It is possible that, during the regularised inversion process, one or more "observations" were "carried along" (with weights of zero) talking no part in the parameter estimation process but, were in fact model predictions of interests. This strategy allows calculation of derivatives of these predictions with respect to all model parameters. If this is the case, these "observations" may have been assigned to an observation group of their own. Unless there is only one such "observation" these observations should not be assigned to an observation group named "predict", as this observation group has special significance when conducting predictive uncertainty analysis. If, however, there is only one observation assigned to the observation group "predict", REGPRED will treat this as the prediction whose task it is for the predictive analysis process to maximise/minimise when it builds the predictive analysis PEST control file.

4.19.4 Running REGPRED

4.19.4.1 REGPRED Dialogue

REGPRED commences execution with the prompt:-

```
Enter name of existing PEST control file:
```

Provide the name of an existing PEST control file containing optimised parameter and observation values; if you omit the *.pst* extension, REGPRED will automatically add this for you. Note that if parameter estimation was carried out using SVD-assist, this should *not* be a super-parameter PEST control file. Rather it should be a base parameter PEST control file (with optimised parameter and observation values as discussed above).

After you have provided the name of this file, and after REGPRED verifies that this file exists, it prompts:-

```
Does it contain optimised parameter and observation values?  [y/n]:
```

If you answer "n" REGPRED's reaction will be swift and decisive:-

```
Then use PARREP and OBSREP to build such a PEST control file.
```

whereupon it will immediately cease execution.

REGPRED's next prompt is:-

```
Use resolution or SVD matrix for null space projection? [r/s]:
```

If the response to the above prompt is "r", REGPRED prompts:-

```
Enter name of corresponding resolution matrix file:
```

As mentioned above, this resolution matrix file should have been produced by running RESPROC followed by RESWRIT after the PEST regularised inversion run through which optimised parameter values (and optimised model outputs) were obtained.

REGPRED next asks:-

```
Enter dimensions of solution space:
```

This question may sometimes be difficult to answer. If regularised inversion had previously been carried out using pure TSVD (with zero Marquardt lambda) in which the truncation limit, rather than ratio of highest to lowest eigenvalue, was set as the cut-off criterion, the answer to the above question is simply the number of eigenvalues specified before truncation. If the eigenvalue ratio was instead employed, then the dimensionality of the solution subspace may be available from the "number of singular values used in solution" specifier recorded in the *svd* file by PEST. Note however that this may vary from iteration to iteration. Use of that pertaining to the last iteration is probably the best idea.

A good way to determine a suitable response to this prompt is to use the MATSVD utility to undertake singular value decomposition of the **R** matrix. Then inspect the list of eigenvalues forthcoming from this exercise. Select the eigenvalue number at which eigenvalues start to become very low. (If the **R** matrix was produced using pure TSVD, or SVD-assist without Tikhonov regularisation and zero lambda, all eigenvalues should be either one or zero.)

If, on the other hand, you had instructed REGPRED to read an SVD matrix instead of a resolution matrix, it will prompt:-

```
Enter name of SVD matrix file:-
```

This must be a file with an extension of "*.svd*" produced during a previous PEST run in which PEST was instructed to use singular value decomposition as a solution device. If the EIGWRITE variable was set to 1 on that run, the **V** matrix of eigenvectors of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ is recorded for every iteration in this file. REGPRED next asks:-

```
Get SVD for which iteration number? <Enter if 1>:
```

If the SVD run was undertaken for the purposes of model calibration, enter the number of the last iteration undertaken. However if PEST was only run for 1 iteration, purely for the sake of obtaining this matrix, then a matrix will be produced only for iteration number 1. See below for a way in which this can be easily achieved based on existing sensitivities even if TSVD had not been used in the previous calibration exercise. REGPRED's next prompt is:-

```
Enter dimensions of solution space:
```

If calibration was undertaken using SVD, this will correspond to the "number of singular values used in solution" pertaining to the last optimisation iteration. However if PEST was run in SVD mode for 1 iteration purely to obtain the **V** matrix, prior to using REGPREG after an SVD-assist run, it will correspond to the number of super parameters employed in the SVD-assist run.

Next, whether REGPRED reads a resolution matrix of SVD file, it prompts for the names of a parameter uncertainty file and an observation uncertainty file:-

```
Enter name of parameter uncertainty file:
Enter name of observation uncertainty file:
```

See documentation of program PARAMERR for the format of these files. The purpose of these files is to supply the C(**p**) and C(**ε**) matrices, the roles of these matrices in the predictive analysis process being described above. Note that if parameters are log-transformed through the parameter estimation process, then pertinent elements of C(**p**) must relate to the logs of these parameters. Note also that if the observation uncertainty file cites a PEST control file (because the weights in this file are the inverse of measurement uncertainties), it must not be the same PEST control file whose name was supplied to REGPRED following the first of its prompts. However it must provide uncertainty information for all observations and prior information equations cited in this PEST control file, except for those which belong to

regularisation observation groups.

REGPRED next prompts for the name of the file that it must write:-

```
Enter name for new predictive analysis PEST control file:
```

This new PEST control file will exclude any regularisation observations and/or prior information cited in the original PEST control file. In this new PEST control file, PEST will be instructed to run in predictive analysis mode. It will also include a "predictive analysis" section containing control variables for the predictive analysis process. One of these variables is NPREDMAXMIN, which informs PEST whether the prediction of interest is to be maximised or minimised. In order to know this, REGPRED prompts:-

```
Maximise or minimise prediction? [a/i]:
```

Then it asks:-

```
Enter value for PDO (<Enter> if 9.0):
```

As is explained in the main PEST manual, PD0 is the objective function constraint imposed through the maximisation/minimisation process. A value of 9.0 is equivalent to three standard deviations (because it is the square of 3) and hence to a two-sided confidence level of about 99.7% for normally distributed variables.

If the PEST control file whose name as supplied to REGPRED contains an observation group named "predict", and if that observation group contains a single observation, REGPRED next prompts:-

```
Treat single member "obs" of observation group "predict"
   as "the prediction" in new PEST control file?  [y/n]:
```

where "obs" is replaced by the observation actually identified in the PEST control file. If the answer to this question is "y", REGPRED then asks:-

```
Incorporate predictive noise in pred. anal. process  [y/n]:
```

See the documentation of alterations to PEST's predictive analyser in another part of this addendum. As explained there, the existence of "predictive noise" can be incorporated into the predictive maximisation/minimisation process undertaken by PEST. If this is done, the PREDNOISE variable in the "predictive analysis" section of the PEST control file must be set to 1, and a non-zero weight must be supplied to the prediction comprising the single member of the observation group "predict". This weight must be the inverse of the standard deviation of predictive noise.

4.19.4.2 The Modified Model

The model run by PEST as part of the predictive analysis process must compute **g** of equation 4.26 on the basis of current model parameter values. REGPRED modifies the model to be able to do this. On the assumption that the original model is a batch or script file (REGPRED will cease execution with an error message if the extension of the model filename is not "*.bat*"), REGPRED adds commands to the beginning of this file as it writes a new model batch file named *regpredbat.bat* which the new PEST control file will employ to run the model.

A template file named *par###.tpl* is added to the PEST input dataset, its corresponding model input file being named *par###.mat*. This contains the vector of current parameters **p**; note that only adjustable parameters are written to this file. The model then runs the VECLOG utility to take the logs of log-transformed parameters, producing a file named *parlog###.mat*. The

transformation matrix file employed by VECLOG (which is written by REGPRED) is named *partran###.mat*.

A difference is then taken between current (transformed) parameter values **p** and optimised parameter values **p**. The latter are stored in file *refpar###.mat*, which is written by REGPRED; note that the values of log-transformed parameters are log-transformed before storage in this file. Vector differencing is undertaken by the MATDIFF utility, the **p**-**p** vector is written to a file named *pardiff###.mat*.

The MATPROD utility is used to compute **g** using equation 4.26. $\mathbf{U}_2^t$ (or $\mathbf{V}_2^t$ if the SVD option is used) from that equation is stored in file *proj###.mat* by REGPRED. **g** itself is written to file *projdiff###.mat* by MATPROD. This file is then read by a new instruction file (written by REGPRED) named *proj###.ins*.

The elements of **g** are assigned to observations named *projdiff1, projdiff2* etc in the new PEST control file written by REGPRED. These in turn are assigned to the observation group *projdiff* to which the covariance matrix C(**g**) (computed using 4.28) is assigned in the "observation groups" section of the new PEST control file written by REGPRED.

It is important to ensure that the VECLOG, MATDIFF and MATPROD executables reside in a directory cited in the PATH environment variable (or reside in the current working directory). If running Parallel PEST, it is important to ensure that these executables are transferred to slave machines.

4.19.5 Some Notes

The following aspects of the input dataset written by REGPRED should be carefully noted.

1. If, in the original PEST control file whose name is supplied to REGPRED, there is an observation group named "predict", and if it contains more than one member, REGPRED will cease execution with an appropriate error message. Alternatively, if there is such a group and it contains only one member, REGRPRED will prompt (as shown above) for permission to treat this one member as the prediction whose task it is for PEST to maximise/minimise in the forthcoming predictive analysis process. If the user does not want this, REGPRED will ask him/her to assign that observation to another observation group. If there is no observation group named "predict" in the original PEST control file REGPRED will create one. It is then the user's task to add an actual prediction to this PEST control file. A new instruction file (or an alteration made to an existing one) will also be required in order to provide PEST with the means to read this new prediction from the model output dataset. REGPRED issues a warning to this effect before finishing execution.
2. As discussed above, observation and prior information weights are not transferred from the original PEST control file to the new one. Rather observation weights are assigned on the basis of observation uncertainties as supplied to REGPRED through an observation uncertainty file. However if any observation or prior information equation is assigned a weight of zero in the original PEST control file, it will also be assigned a weight of zero in the new PEST control file as this is taken as an indication that the user is not interested in this observation.
3. All regularisation observations and prior information equations are eliminated from the new PEST control file written by REGPRED. On many occasions of PEST usage regularisation constraints are limited solely to prior information. The elimination of these prior information equations from the new PEST control file has no adverse

consequences. However where observations, rather than prior information equations, are eliminated from the PEST control file because they belong to a regularisation group, pertinent instruction files comprising the PEST input dataset need to be deleted or modified. It is the user's responsibility to undertake this task (REGPRED reminds the user of this before ceasing execution).

4. If a "singular value decomposition" section exists in the original PEST control file, it is removed.
5. Irrespective of the NOPTMAX (current number of iterations) setting in the original PEST control file, a NOPTMAX setting of 50 is provided to the new PEST control file. (This can prevent the user from severe disappointment if, on returning to his/her computer next morning, he she finds that PEST has run the model only once in order to calculate the objective function, or that PEST ran for only one iteration in order to calculate parameter sensitivities because he/she forgot to alter the NOPTMAX setting.)
6. Predictive analysis control variables written to the "predictive analysis" section of the new PEST control file by REGPRED are conservative. A line search is instigated, the control variables for this line search being such as to sample the line densely enough to detect the existence of a possible fall in the objective function followed by a rise before intersection of the PD0 contour. Also, convergence criteria are tight. If these settings are not suitable, the user should alter them him/herself.

When PEST is run on the basis of the new PEST control file (it may be necessary to add the prediction to this file first as discussed above) the initial objective function should be zero. Occasionally PEST may cease execution after the first iteration, saying that the "phi gradient is zero". If this is the case, perturb the initial value of one of the parameters by a slight amount and re-commence PEST execution.

4.19.6 Using the SVD Matrix Option

As mentioned above, use of the SVD-matrix option rather than the resolution matrix option, can be advantageous where the calibration process was undertaken using SVD-assist based on PEST-determined super parameters. In this case, use of a $\mathbf{V}_2$ matrix in place of $\mathbf{U}_2^t$ **(I – R)** ensures that the second term of equation 4.23 makes no contribution to the null space component of computed error variance. To obtain an SVD file (which must be based on base parameters) under these circumstances, undertake the following steps. Note that these steps assume that a JCO file already exists for the base parameter file, and that PEST-determined super parameters, rather than user-supplied super parameters, were employed in the calibration process.

1. Copy the base parameter pest control file to another file.
2. Set NUMLAM to 1 and RLAMBDA1 to zero in that file.
3. Remove all regularisation observations and prior information.
4. Add a "singular value decomposition" section to this file. Be sure to set EIGWRITE to 1.
5. Set NOPTMAX to 1.
6. Use the JCO2JCO utility to create a JCO file corresponding to this new PEST control file from the original JCO file pertaining to the base PEST control file.

7. Start PEST on the basis of the new PEST control file using the "/i" switch; when prompted for the name of a JCO file, supply the name of the JCO file just created by JCO2JCO. PEST will then undertake three model runs – one at the start of the process, one to test an upgrade vector, and one on the basis of "best fit parameters". However all model runs normally employed for calculation of the Jacobian matrix will be dispensed with as parameter sensitivities are read from the existing JCO file.

4.20 PREDUNC1

4.20.1 Theory

PREDUNC1 computes predictive uncertainty conditional upon the acquisition of data comprising the calibration dataset. It can be used interchangeably with PREDVAR1.

Suppose that the innate variability of system properties **p** is described by the covariance matrix C(**p**). Suppose also that the sensitivity of a prediction *s* to system parameters **p** is represented by the vector **y**. Thus if the model is linear, the relationship between the prediction *s* and parameters **p** is described by the equation (ignoring offsets):-

$$s = \mathbf{y}^t\mathbf{p} \tag{4.29}$$

The uncertainty of that prediction can be characterised by a variance σ^2_s computed as:-

$$\sigma^2_s = \mathbf{y}^t C(\mathbf{p})\mathbf{y} \tag{4.30}$$

Suppose that the vector **h** represents a calibration dataset whose relationship to parameters is given by:-

$$\mathbf{h} = \mathbf{X}\mathbf{p} + \boldsymbol{\varepsilon} \tag{4.31}$$

where **X** is a sensitivity matrix. Combining (4.31) and (4.29) we have:-

$$\begin{bmatrix} s \\ \mathbf{h} \end{bmatrix} = \begin{bmatrix} \mathbf{y}^t & \mathbf{0} \\ \mathbf{X} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \boldsymbol{\varepsilon} \end{bmatrix} \tag{4.32}$$

and thus, using standard matrix relationships for propagation of covariance:-

$$\begin{aligned} C\left(\begin{bmatrix} s \\ \mathbf{h} \end{bmatrix}\right) &= \begin{bmatrix} \mathbf{y}^t & 0 \\ \mathbf{X} & \mathbf{I} \end{bmatrix} \begin{bmatrix} C(\mathbf{p}) & 0 \\ 0 & C(\boldsymbol{\varepsilon}) \end{bmatrix} \begin{bmatrix} \mathbf{y} & \mathbf{X}^t \\ 0 & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{y}^t C(\mathbf{p})\mathbf{y} & \mathbf{y}^t C(\mathbf{p})\mathbf{X}^t \\ \mathbf{X}C(\mathbf{p})\mathbf{y} & \mathbf{X}C(\mathbf{p})\mathbf{X}^t + C(\boldsymbol{\varepsilon}) \end{bmatrix} \end{aligned} \tag{4.33}$$

Now suppose that an arbitrary vector **x** is partitioned into two separate parts $\mathbf{x}_1$ and $\mathbf{x}_2$. That is:-

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \tag{4.34}$$

Let C(**x**), the covariance matrix of **x**, be correspondingly partitioned as:-

$$C(\mathbf{x}) = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{4.35}$$

Suppose further that the elements of $\mathbf{x}_2$ become known. Then, if there is correlation between at least some members of $\mathbf{x}_2$ and some members of $\mathbf{x}_1$ (resulting in non-null $\mathbf{C}_{12}$ and $\mathbf{C}_{21}$ submatrices), the conditioned $\mathbf{C}_{11}$matrix $\mathbf{C}'_{11}$ is calculable as:-

$$\mathbf{C}'_{11} = \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}^{-1}{}_{22}\mathbf{C}_{21} \tag{4.36}$$

provided $\mathbf{x}$ has a multi-Gaussian probability distribution. Applying this to equation 4.33, the conditional variance of our prediction *s* given the acquisition of calibration data $\mathbf{h}$, is:-

$$\sigma_s^2 = \mathbf{y}^t\mathrm{C}(\mathbf{p})\mathbf{y} - \mathbf{y}^t\mathrm{C}(\mathbf{p})\mathbf{X}^t\left[\mathbf{X}\mathrm{C}(\mathbf{p})\mathbf{X}^t + \mathrm{C}(\boldsymbol{\varepsilon})\right]^{-1}\mathbf{X}\mathrm{C}(\mathbf{p})\mathbf{y} \tag{4.37}$$

PREDUNC1 computes σ^2_s of equation 4.37. Note that $\mathbf{y}^t\mathrm{C}(\mathbf{p})\mathbf{y}$ (the first term of equation 4.37), is the pre-calibration uncertainty of the prediction *s*; PREDUNC1 computes this as well.

Where observation numbers are large, the $\mathbf{X}\mathrm{C}(\mathbf{p})\mathbf{X}^t + \mathrm{C}(\boldsymbol{\varepsilon})$ matrix of equation (4.37) will be difficult to invert because of its size; its dimension are $n \times n$ where n is the number of observations comprising the calibration dataset. However equation (4.37) can be altered to form that is more easily accomodated numerically if parameter numbers are not too large. Before demonstrating how this is done, a handy matrix transformation will be proved.

Let us start with the obvious identity:

$$\mathbf{B}^t\mathbf{D}^{-1}\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{B}^t = \mathbf{B}^t\mathbf{D}^{-1}\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{B}^t \tag{4.38}$$

Then, noting that:

$$\mathbf{D}^{-1}\mathbf{D} = \mathbf{I} \tag{4.39}$$

and

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \tag{4.40}$$

This can be re-written as:

$$\mathbf{B}^t\mathbf{D}^{-1}\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{B}^t\mathbf{D}^{-1}\mathbf{D} = \mathbf{B}^t\mathbf{D}^{-1}\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{A}^{-1}\mathbf{A}\mathbf{B}^t \tag{4.41}$$

Thus:

$$\mathbf{B}^t\mathbf{D}^{-1}(\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{D}) = (\mathbf{B}^t\mathbf{D}^{-1}\mathbf{B} + \mathbf{A}^{-1})\mathbf{A}\mathbf{B}^t \tag{4.42}$$

By premultiply both sides of (4.42) by $(\mathbf{B}^t\mathbf{D}^{-1}\mathbf{B} + \mathbf{A}^{-1})^{-1}$ and postmultiply both sides by $(\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{D})^{-1}$ we then obtain:

$$(\mathbf{B}^t\mathbf{D}^{-1}\mathbf{B} + \mathbf{A}^{-1})^{-1}\mathbf{B}^t\mathbf{D}^{-1} = \mathbf{A}\mathbf{B}^t(\mathbf{B}\mathbf{A}\mathbf{B}^t + \mathbf{D})^{-1} \tag{4.43}$$

Now from (4.37):

$$\sigma^2_s = \mathbf{y}^t\{\mathrm{C}(\mathbf{p}) - \mathrm{C}(\mathbf{p})\mathbf{X}^t[\mathbf{X}\mathrm{C}(\mathbf{p})\mathbf{X}^t + \mathrm{C}(\boldsymbol{\varepsilon})]^{-1}\mathbf{X}\mathrm{C}(\mathbf{p})\}\mathbf{y} \tag{4.44}$$

which, using the identity exressed by (4.42) becomes:

$$\sigma^2_s = \mathbf{y}^t\{\mathrm{C}(\mathbf{p}) - [\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})]^{-1}\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X}\mathrm{C}(\mathbf{p})\}\mathbf{y} \tag{4.45}$$

$$= \mathbf{y}^t\{(\mathbf{I} - [\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})]^{-1}\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X})\mathrm{C}(\mathbf{p})\}\mathbf{y} \tag{4.46}$$

From the obvious identity:

$$\mathbf{I} = [\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})]^{-1}[\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})] \tag{4.47}$$

it follows from (4.46) that:

$$\sigma^2_s = \mathbf{y}^t\{([\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})]^{-1}\,[\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})] - [\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathrm{C}^{-1}(\mathbf{p})]^{-1}\mathbf{X}^t\mathrm{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X})\mathrm{C}(\mathbf{p})\}\mathbf{y}$$

$$= \mathbf{y}^t\{([\mathbf{X}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathbf{C}^{-1}(\mathbf{p})]^{-1}\,[\mathbf{X}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathbf{C}^{-1}(\mathbf{p}) - \mathbf{X}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X})\mathbf{C}(\mathbf{p})\}\mathbf{y} \qquad (4.48)$$

from which it follows that:

$$\sigma^2_s = \mathbf{y}^t[\mathbf{X}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathbf{C}^{-1}(\mathbf{p})]^{-1}\mathbf{y} \qquad (4.49)$$

The matrix $\mathbf{X}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathbf{C}^{-1}(\mathbf{p})$ requiring inversion in equation (4.49) has dimensions equal to the number of elements of **p**, i.e. the number of parameters.

4.20.2 Using PREDUNC1

Use of PREDUNC1 is very similar to that of PREDVAR1. However a list of singular values is not required for computation of σ^2_s using either of equations (4.37) or (4.49); hence singular values are not requested.

Like PREDVAR1, PREDUNC1 commences execution with the prompts:-

```
Enter name of PEST control file:
```

and

```
Enter observation reference variance:
```

As is done by PREDVAR1, PREDUNC1 obtains statistics of measurement noise (i.e. the C(**ε**) matrix) from the user-nominated PEST control file. Where measurement weights are employed in this file, PREDUNC1 assumes that the square of each such weight is proportional to the inverse of the variance of the noise associated with each respective measurement, the proportionality constant being the reference variance σ^2_r. Thus the actual standard deviation of measurement noise is presumed to be σ_r times the inverse of the weight of each measurement, where σ_r is the square root of the reference variance as supplied by the user. Where a covariance matrix is supplied for some or all observations, this same reference variance is assumed to apply.

PREDUNC1's next prompt is:

```
Enter name of parameter uncertainty file:
```

in response to which the name of an appropriate file containing the C(**p**) matrix must be supplied. PREDUNC1's next prompt is:-

```
Enter name of predictive sensitivity matrix file:
```

The response to this prompt must be the name of a file written by a program such as JROW2VEC which lists the sensitivity of a prediction to each adjustable parameter. (Note that the ordering of these parameters in the predictive sensitivity file does not need to be the same as that in the PEST control file; nor does it matter if extra parameters are cited in this file.)

Next PREDUNC1 asks:

```
Use which version of linear predictive uncertainty equation:-
     if version optimized for small number of parameters   - enter 1
     if version optimized for small number of observations - enter 2
 Enter your choice:
```

The "version optimized for small number of parameters" is equation (4.49); the version optimized for small number of observations is (4.37).

PREDUNC1 next computes the pre- and post-calibration uncertainty of the prediction, and writes these to the screen.

Unlike the PREDVAR utilities, PREDUNC does not compute the separate contributions to predictive error variance made by the calibration solution and null spaces, for no singular value decomposition is undertaken of the weighted sensitivity matrix. However contributions made by these spaces can be computed in an indirect way by increasing the weights assigned to all measurements. When measurement weights are very high, the resulting predictive uncertainty is attributable solely to nonuniqueness of solution of the inverse problem; that is, it is a direct outcome of the existence and size of the calibration null space.

4.21 PREDUNC4

PREDUNC4 is to PREDUNC1 what PREDVAR4 is to PREDVAR1. That is, it computes the predictive uncertainty where groups of parameters are sequentially removed from the parameter estimation process, the names of these parameters being supplied in a "parameter type list file". However, unlike PREDVAR4, it does not prompt for a singular value list file, for the means by which predictive uncertainty is calculated is the same as that employed by PREDUNC1, this not requiring that singular value decomposition of the weighted sensitivity matrix be undertaken.

The user is therefore referred to documentation of PREDVAR4 for usage details of PREDUNC4.

4.22 PREDUNC5

PREDUNC5 is to PREDUNC1 what PREDVAR5 is to PREDVAR1. That is, it computes the predictive uncertainty where groups of measurements are sequentially added or subtracted from the calibration dataset, the names of these observations being supplied in an "observation type list file". However, unlike PREDVAR5, it does not prompt for a singular value list file, for the means by which predictive uncertainty is calculated is the same as that employed by PREDUNC1, this not requiring that singular value decomposition of the weighted sensitivity matrix be undertaken.

The user is therefore referred to documentation of PREDVAR5 for usage details of PREDUNC5.

4.23 PREDUNC6

PREDUNC6 performs a similar function to PREDUNC1 in that it computes pre- and post-calibration uncertainty. Unlike PREDUNC4 and PREDUNC5, it does not compute any "value-added" information such as contributions made to uncertainty by different parameters, or alterations to predictive uncertainty promulgated through inclusion or exclusion of observations from the calibration process.

In contrast to PREDUNC1, PREDUNC6 computes pre-and post-calibration uncertainties for multiple predictions. It records the outcomes of its calculations in a tabular data file that is readily amenable to processing using other software.

Like PREDUNC1, PREDUNC6 issues the following prompts; typical responses are also shown below.

```
Enter name of PEST control file: case13.pst
Enter observation reference variance: 1.0

Enter name of parameter uncertainty file: param.unc
```

PREDUNC6's next prompt is:

```
Enter name of predictive sensitivity JCO file:
```

This file must be a Jacobian matrix file written by PEST. Hence it should contain the sensitivity of many "observations" cited in its complementary PEST control file, to all adjustable parameters cited in that file. PREDUNC6 assumes that every "observation" is in fact a prediction whose uncertainty is to be computed. For this to be possible, the Jacobian matrix file should feature the same parameters as those that are featured in the PEST control file on which notional model calibration is based, that is the PEST control file whose name is supplied in response to the first of PREDUNC6's prompts (and for which a complimentary JCO file must exist). However it does not matter if the predictive sensitivity JCO file features more parameters than this, or if the parameters are represented in a different order; PREDUNC6 will undertake the necessary adjustment and re-ordering.

Next PREDUNC6 prompts for the name of the file that it must write:

```
Enter name for output uncertainty table file:
```

Upon completion of PREDUNC6 execution, this file will contain a listing of all predictions named in the predictive sensitivity JCO file, as well as the pre-calibration and post-calibration uncertainty, and uncertainty variance, associated with each (the latter is the square of the former).

PREDUNC6's final prompt is:

```
Use which version of linear predictive uncertainty equation:-
     if version optimized for small number of parameters   - enter 1
     if version optimized for small number of observations - enter 2
 Enter your choice:
```

See documentation of PREDUNC1 for details of these options.

Note the following:

1. Don't forget the extremely useful JCO2JCO utility. This allows you to modify a PEST control file (for example by removing observations), and to then obtain a complementary JCO file. This can be very useful in constructing the predictive sensitivity JCO file required by PREDUNC6.
2. Parameters cited in the PEST control file on which the predictive sensitivity JCO file is based, must be logarithmically transformed (or not) in the same way as in the PEST control file on which notional calibration is based. Similarly, if a parameter is a parent parameter to tied parameters in the latter file, it must perform the same role in the former file. These measures ensure consistency of sensitivities.

4.24 GENLINPRED

4.24.1 General

GENLINPRED stands for "general linear predictive uncertainty/error analyser". It is actually a driver program which runs a number of other PEST utilities, these being SCALEPAR, JROW2VEC, SUPCALC, IDENTPAR, PREDUNC1, PREDUNC4, PREDUNC5, PREDVAR1, PREDVAR4 and PREDVAR5. As such, it carries out a variety of tasks related to assessment of the uncertainty and/or error variance of a parameter or prediction, collecting the information that it gathers into a single output file. Tables within this file are readily pasted into Microsoft EXCEL, or any other graphing program, for graphical analysis if

desired.

Because it is a driver program, and because it therefore runs other programs to carry out various computational tasks, some of these tasks are repeated by the different programs (for example the task of undertaking singular value decomposition of a weighted sensitivity matrix). Thus, from a computation point of view, using GENLINPRED to conduct error/uncertainty analysis is inefficient. However from a user's point of view its use can be efficient, for it performs a multitude of related tasks based on minimal user keyboard input and/or input file preparation.

4.24.2 Tasks Carried out by GENLINPRED

GENLINPRED carries out some or all (as requested by the user) of the following tasks.

1. It optionally uses SUPCALC to compute the optimum dimensionality of the calibration solution and null spaces for a particular parameter estimation problem.
2. On the basis of this subdivision (or using calibration solution and null space dimensionalities supplied by the user), it can compute the identifiability of each parameter (using the IDENTPAR utility) and the relative error variance reduction of each parameter (using PREDVAR1A). It can also compute the relative uncertainty variance reduction of each parameter using the PREDUNC1 utility; note that this latter quantity does not require subdivision of parameter space into solution and null subspaces.
3. If requested GENLINPRED computes the solution space and null space components of the total error variance of a nominated prediction (which may in fact be a single parameter) at different singular value truncation levels. This allows the user to graph the dependence of these quantities on the number of pre-truncation singular values employed in calibration of a model. It uses PREDVAR1 for this purpose.
4. Pre- and post-calibration uncertainties of a nominated prediction can be computed using the PREDUNC1 utility.
5. The contributions to the pre- and post-calibration error variance and/or uncertainty of a nominated prediction (parameter) made by different parameter groups, or by different individual parameters, can be computed using the PREDVAR4 and/or PREDUNC4 utilities.
6. The worth of different observation groups, or of different individual observations, in lowering the post-calibration error variance and/or uncertainty of a nominated prediction (parameter) can be computed using PREDVAR5 and/or PREDUNC5 by selectively removing those observation (groups) from the total calibration dataset and monitoring the rise in predictive error variance (uncertainty variance) thereby incurred.
7. The worth of different observation groups, or of different individual observations, in lowering the post-calibration error variance and/or uncertainty of a nominated prediction (parameter) can be computed using PREDVAR5 and/or PREDUNC5 by selectively adding those observation (groups) to a null calibration dataset and monitoring the fall in predictive error variance (uncertainty variance) thereby incurred.

All operations are carried out on a PEST input dataset that is modified by GENLINPRED from that supplied by the user, in that all parameters are scaled by their pre-calibration

variabilities. Where calibration is actually, or notionally, implemented using truncated SVD, this operation results in lower predictive error variance than where unscaled parameters are estimated through the calibration process.

4.24.3 Predictive Error Variance and Predictive Uncertainty

As is explained in documentation of the PREDVAR and PREDUNC suite of utilities, the former calculate the error variance of a particular prediction (which may also be a parameter), whereas the later calculate the uncertainty of that prediction (parameter). Uncertainty is an intrinsic quality of the parameters and of the calibration dataset, and is obtained by conditioning a pre-calibration parameter uncertainty matrix on the basis of observations comprising the calibration dataset. On the other hand, "error" is a concept that is associated with the notion of calibration, and quantifies the extent to which a prediction made by a calibrated model may be wrong. The PREDVAR utility suite notionally implements model calibration on the basis of truncated singular value decomposition; truncation takes place at that singular value at which the error variance of the prediction of interest is minimized.

It is important to note that even though GENLINPRED (and the utility programs that it runs) computes predictive uncertainty and predictive error variance, it does not actually make a prediction; nor does it calibrate a model. Its calculations are made on the basis of sensitivities alone, and no parameter adjustment actually takes place. Moreover, a model does not need to be actually calibrated for GENLINPRED to be employed for calculation of quantities that are salient to model calibration, parameterisation, and predictive uncertainty analysis. Sensitivities calculated on the basis of current parameter values are employed, whether or not those parameter values result in a calibrated model. Naturally, the results of GENLINPRED analysis may show some degree of parameter-dependence for seriously nonlinear models where sensitivities are strong functions of parameter values. However experience has demonstrated that broad outcomes of linear analysis are robust on most occasions. Thus, for example, if a particular parameter group is identified as making a large contribution to the uncertainty of a particular prediction relative to that made by other parameter groups on the basis of current parameter values, that conclusion is likely to be robust. However the actual number calculated for that contribution is likely to change with parameter values.

4.24.4 The C(p) Matrix

As is discussed elsewhere in this manual, highly-parameterized error/uncertainty analysis requires that the user provide a covariance matrix of innate parameter variability. This can also be viewed as a covariance matrix of pre-calibration parameter uncertainty. As such, this matrix provides a statistical encapsulation of what is known, and of what is unknown, about system properties before an attempt is made to refine that knowledge through matching model outputs to historical observations of system state. Lack of knowledge of system parameters is expressed by the fact that the C(**p**) matrix has non-zero diagonal elements, thereby demonstrating that parameter values are only approximately known. Knowledge is expressed by the fact that these diagonal elements are finite, and that non-zero off-diagonal elements may depict a propensity for spatial correlation of heterogeneous parameter fields.

The C(**p**) matrix is supplied through an uncertainty file. The format of this file is discussed elsewhere in this manual. This file can be easily prepared using a text editor. Where pilot points parameterisation is employed, assistance in its preparation can be obtained through the PPCOV and PARCOV utilities supplied with the Groundwater Data Utility suite.

It is important to note that if a parameter is designated as log-transformed in the PEST control

file supplied to GENLINPRED, its pre-calibration uncertainty, as provided in the uncertainty file, must pertain to the log (to base 10) of the parameter

4.24.5 Observations and Predictions

Use of GENLINPRED requires that a PEST control file and corresponding JCO file be provided to it. The latter can be obtained from the former by running PEST with the NOPTMAX termination control variable set to -1 or -2. Observations cited within the "observation data" section of the PEST control file comprise the calibration dataset. These provide the basis for estimation of parameters through the notional truncated SVD calibration exercise undertaken by the PREDVAR suite of programs; they provide the basis for conditioning of pre-calibration parameter uncertainty undertaken using the PREDUNC suite of programs.

The prediction whose uncertainty and error variance is analysed by GENLINPRED can be any function of the adjustable parameters cited in the supplied PEST control file. For a linear model, this function is represented by its sensitivities to all adjustable parameters. These sensitivities may be supplied by the user through a "sensitivity file" (which must adopt PEST matrix file format). Alternatively they may comprise one row of a Jacobian matrix, either the Jacobian matrix that complements the PEST control file, or another Jacobian matrix altogether, that was produced on the basis of a PEST control file that was built specifically for obtaining the sensitivity of one or a number of predictions to the same adjustable parameters as are employed in the calibration process. A third possibility is that the "prediction" may actually be a parameter. In this case GENLINPRED writes the prediction sensitivity file itself; sensitivities of this prediction to all parameters, except for the one in question, are zero.

A particularly easy way to implement predictive error and uncertainty analysis is to build a PEST control file that includes not only observations that are employed in the calibration process, but also one or a number of predictions that are to be subjected to error/uncertainty analysis. When PEST is then run with NOPTMAX set to -1 or -2, sensitivities of these predictions to all adjustable parameters are automatically computed, along with the sensitivities of observations comprising the calibration dataset. If the "prediction observations" are assigned weights of zero, they do not actually take part in the parameter estimation process; they are simply "carried" in that process for the purpose of obtaining sensitivities. When using GENLINPRED, the user then informs it that predictive sensitivities reside in the JCO file that complements the PEST control file that is supplied to it.

4.24.6 Making GENLINPRED Easier to Use

As it requires more information than can be provided through the command line, GENLINPRED gathers information from the user through the user's responses to a series of prompts. Unfortunately there are many prompts, and it is easy to make a mistake in responding to these prompts. Because of this GENLINPRED provides backtracking capabilities. If you respond to any prompt by simply pressing "e" (for "escape") followed by <Enter>, GENLINPRED will take you back to its previous prompt so that you may provide an alternative response to this prompt if you wish. This process can continue right back to GENLINPRED's first prompt.

In order to reduce the need to provide responses for a large number of prompts, GENLINPRED provides the user with two options for prompt sequences. As will be seen shortly, through one of its early prompts GENLINPRED provides the user with the option of

receiving an abbreviated set of prompts. If this option is selected, the user is not asked whether he/she wishes to undertake parameter estimability analysis; instead it is assumed that this is not required. Furthermore, when undertaking parameter/predictive error/uncertainty analysis, no prompts are in fact issued for error analysis, so that only uncertainty analysis is undertaken.

As a further measure for enhancing ease of use, a default response is provided for most of GENLINPRED's prompts; by pressing the <Enter> key, the default is accepted. The user should note, however, that while responding with a simple press of the <Enter> key may be the easiest way to respond to a prompt, it is not always the correct way. This applies in particular to prompts regarding observation weights and the use of parameter bounds as a measure of pre-calibration parameter uncertainty.

As a final means to expedite GENLINPRED user interaction, GENLINPRED writes an optional "response file". This records the user's responses to each of its prompts. Suppose that this file is named *genlinpred.rsp* (it is the user's choice). Then after it has been run on the basis of keyboard input supplied by the user in response to each of its individual prompts, it can then be run again using the command:

```
genlinpred < genlinpred.rsp
```

to produce identical output. Alternatively, one or more responses to GENLINPRED's prompts can be altered through editing of this file before issuing the above command. Because GENLINPRED labels its prompts when writing the response file (see the figure below) this is easy to do. However if you do this, it is important to note the following.

1. The number and nature of GENLINPRED's prompts depend on answers to previous prompts. Hence if, for example, you alter a "y" to a "n" in the response file, this may invalidate the file, as the ensuing questions may be different, or may not be asked at all. However responses to prompts for items such as filenames, number of singular values, the name of a prediction or parameter etc can be altered with impunity.
2. When reading its response file GENLINPRED ignores all characters following the "!" character. This character should not be removed from any line of the file. However it, together with ensuing text, can be moved to the right to make room for a longer response to a particular prompt (this will apply only to filenames) if this is required.

```
! GENLINPRED response file. Beware of altering single letter responses as ensuing
GENLINPRED prompts may be different
f                    ! abbreviated or full input?
temp1.pst            ! PEST control file
b                    ! bounds or uncertainty file for parameter uncertainties?
y                    ! are weights the inverse of measurement uncertainty?
genlinpred.out       ! GENLINPRED output file
y                    ! perform global parameter estimability analysis?
y                    ! compute parameter identifiabilities?
y                    ! compute relative parameter error reduction?
y                    ! use SUPCALC to estimate solution space dimensionality?
y                    ! compute relative parameter uncertainty reduction?
y                    ! perform comprehensive analysis of prediction or parameter?
ar10                 ! name of prediction or parameter to analyse
ar10.vec             ! file to read predictive sensitivities or "p" for parameter
y                    ! compute solution/null space contributions to predictive error?
y                    ! compute predictive uncertainty?
y                    ! compute parameter contributions to parameter or predictive error?
y                    ! compute parameter contributions to uncertainty?
g                    ! for individual parameters or parameter groups?
y                    ! compute observation worth with respect to error?
y                    ! compute observation worth with respect to uncertainty?
g                    ! for individual observations or for observation groups?
y                    ! over-ride SUPCALC calculation of solution space dimensions?
e                    ! escape
y                    ! over-ride SUPCALC calculation of solution space dimensions?
7                    ! new solution space dimensions
```

Example of a GENLINPRED response file.

4.24.7 Using GENLINPRED

Because GENLINPRED runs other programs of the PEST suite, it is important to ensure that the executable files for these programs reside in a directory that is cited in the PATH environment variable. It is also important to ensure that versions of these programs are no earlier than 11.12.

GENLINPRED begins execution with the prompt:

```
Enter name of response file (<Enter> if none):
```

In response to this prompt, provide the name of the file to which ensuing GENLINPRED prompts, together with your responses to these prompts, will be recorded. Alternatively simply press the <Enter> key. Next it asks:

```
Use abbreviated or full input? [a/f] (<Enter if "f"):
```

If the "a" option is chosen, a subset of the following set of prompts will be offered, as is explained above.

Having dealt with the preliminaries, GENLINPRED now gets down to business. It asks:

```
Enter name of PEST control file:
```

Supply the name of a PEST control file which meets the following specifications.

1. It must contain no prior information.
2. It must instruct PEST to run in estimation mode.
3. A JCO file corresponding to this file must be present in the same directory as that in which the PEST control file resides.

If any of these conditions are violated GENLINPRED will cease execution with an appropriate error message.

GENLINPRED next asks the user for a C(**p**) matrix. Two options are available, as evinced by

the following prompt.

```
Use bounds or uncert file for param uncertainties  [b/u] <Enter> if "b":
```

If the user responds with "b", GENLINPRED builds a C(**p**) matrix itself, this being a diagonal matrix which thereby assumes statistical independence of all adjustable parameters. The standard deviation of each parameter is obtained by dividing the difference between its upper and lower bounds (as represented in the PEST control file) by 4, this strategy being based on the assumptions that:

1. parameters are normally distributed, and
2. their upper and lower bounds approximately demarcate their 95% confidence intervals.

On the other hand, if the user's response to the above prompt is "u", GENLINPRED prompts for the name of an uncertainty file. The prompt is:

```
Enter name of parameter uncertainty file:
```

Calculation of predictive error/uncertainty requires knowledge of the statistics of measurement noise. GENLINPRED assumes that the weight provided for each observation in the PEST control file is inversely proportional to the uncertainty associated with each field measurement. First if asks:

```
Are weights the inverse of measurement uncertainty?  [y/n] <Enter> if "y":
```

A response of "y" signifies a proportionality constant of 1.0. However, if the user's response to the above prompt is "n", GENLINPRED asks:

```
Enter factor for weights to make this so:
```

Provide a factor here by which all weights should be multiplied in order for each of them to thereby equal the uncertainty associated with the measurement to which it is assigned. This factor will also be applied to the inverse of any observation covariance matrices supplied in the "observation groups" section of the PEST control file.

Conceptually, weights are equal to the inverse of measurement uncertainties when the calibration objective function is roughly equal to the number of non-zero-weighted observations comprising the calibration dataset (for then each squared weighted residual is, on average, approximately equal to 1.0). Practically, however, weights supplied to GENLINPRED may need to be lower than this (suggesting higher measurement noise), to account for the fact that the presence of structural error within the measurement dataset (which always shows a high degree of temporal and/or spatial correlation) diminishes its information content to a greater degree than the presence of noise which shows little or no spatial/temporal correlation. This is further discussed below.

Next GENLINPRED prompts for the name of its output file. This is the file to which the outcomes of all of GENLINPRED's analyses will be written. The prompt is:

```
Enter name for output file <Enter> if genlinpred.out:
```

GENLINPRED's next prompt is:

```
Perform global parameter estimability analysis? [y/n] <Enter> if "n":
```

This type of analysis pertains only to parameters. If the response to the above prompt is "y" the user is given the option of undertaking the following types of analysis.

```
Compute parameter identifiabilities? [y/n] <Enter> if "y":
Compute relative parameter error reduction? [y/n] <Enter> if "n":
```

```
Use SUPCALC to estimate soln space dimensions? [y/n] <Enter> if "y":
Compute relative parameter uncertainty reduction? [y/n] <Enter> if "n":
```

Note that the third of the above questions is asked only if the response to either of the first two prompts is "y", for calculation of both parameter identifiability and relative parameter error reduction requires knowledge of the dimensions of the calibration solution (and hence null) spaces. These can be evaluated by SUPCALC; if so, the user has the ability to over-ride SUPCALC's computation (see below). Alternatively, the user may supply the dimensions of the calibration solution space him/herself and dispense with the running of SUPCALC altogether. Hence if the response to the third of the above prompts is "n", GENLINPRED asks.

```
Enter solution space dimensionality?
```

in response to which a number greater than zero and less than or equal to the number of adjustable parameters featured in the PEST control file must be supplied.

GENLINPRED's next prompt is:

```
Perform comprehensive analysis of a prediction/param? [y/n] <Enter> if "y":
```

If the response to this prompt is "no", GENLINPRED commences work (see below). Alternatively, if the response to this prompt is "y", GENLINPRED then inquires:

```
Enter name of prediction/parameter to analyse:
```

If the name of a prediction is supplied in response to the above prompt, GENLINPRED asks:

```
Enter file to read its sensitivities ["p" if a parameter]:
```

If the user wishes to analyse the error/uncertainty of a particular parameter as if it were a prediction, then that parameter's name should be provided in response to the preceding prompt. The fact that this is a parameter must then be indicated by responding to the current prompt with "p". Otherwise, provide the name of the file in which the sensitivities of the prediction can be found. If the extension of the supplied filename is "*jco*", GENLINPRED assumes that the file is a JCO file; it will then use the JROW2VEC utility to extract sensitivities from the pertinent row of this file, re-writing these in PEST matrix file format. Alternatively, if any other filename extension is supplied, GENLINPRED will assume that the predictive sensitivity file is already in PEST matrix file format.

The user is then presented with options for prediction/parameter analysis. These commence with:

```
Compute total predictive error and soln/null space contribs?  [y/n]:
Compute total predictive uncertainty?  [y/n]:
```

An affirmative response to the first of the above prompts will result in GENLINPRED running PREDVAR1 to compute predictive error variance at many different singular value truncation levels, this providing the information through which graphs of solution and null space contributions to predictive error variance versus number of singular values used in estimation of parameters may be computed. An affirmative response to the second of the above prompts will cause GENLINPRED to employ PREDUNC1 to compute the pre- and post-calibration uncertainties of the chosen prediction or parameter.

GENLINPRED then asks:

```
Compute parameter contributions to error? [y/n] <Enter> if "n":
Compute parameter contributions to uncertainty? [y/n] <Enter> if "y":
```

and, if the response to either of the above prompts is "y":

```
 For indiv parameters or for parameter groups? [i/g] <Enter> if "g":
```

PREDVAR4 and PREDUNC4 are employed for calculation of parameter contributions to predictive error and uncertainty variance respectively. Contributions can be calculated for either individual parameters or for groups of parameters. The former is recommended, for the latter may take a long time. The names of parameter groups are read from the "parameter groups" section of the PEST control file. The group to which each parameter belongs is cited in the "parameter data" section of the PEST control file. (Note that if a parameter group contains no adjustable parameters, it is not featured in this analysis.)

Next GENLINPRED asks:

```
 Compute observation worth wrt error? [y/n] <Enter> if "n":
 Compute observation worth wrt uncertainty? [y/n] <Enter> if "y":
```

and, if the response to either of the above prompts is "y":

```
 For indiv observations or for observation groups? [i/g] <Enter> if "g":
```

PREDVAR5 and PREDUNC5 are employed for calculation of the worth of individual observations, or groups of observations (the option being selected by the user in response to the last of the above prompts – the "groups" option is recommended as the "individual" option may require too much computation). Two means of calculating observation worth are provided by these programs. One method is to compute the increase in predictive error/uncertainty accrued through omitting the nominated observation (group) from the calibration dataset; the other is to compute the decrease in pre-calibration error/uncertainty incurred through having that observation (group) as the sole member of the calibration dataset.

GENLINPRED then gets down to work. Regardless of the user's selected processing options, GENLINPRED first runs SCALEPAR to create a PEST input dataset based on scaled parameters. Then it may run SUPCALC to compute an appropriate dimensionality for the calibration solution and null spaces. If so, it writes the outcome of this calculation to the screen and asks the user whether he/she would like to accept this, or override it. The prompt is:

```
SUPCALC has recommended the use of N solution space dimensions
 for computation of parameter identifiability and relative
 error reduction.


 Do you wish to over-ride this? [y/n] <Enter> if "n":
```

If the response is "y", an appropriate solution space dimensionality must be provided (in response to a GENLINPRED prompt requesting this); this is further discussed below. Note that the dimensionality of the solution and null spaces only features in computation of identifiability and relative parameter error reduction. Computation of relative parameter uncertainty reduction, and of all prediction-related quantities, is independent of this choice. Where uncertainty analysis is undertaken (by PREDUNC programs) no formal subdivision of parameter space into solution and null spaces is required. Where error variance analysis is undertaken (by members of the PREDVAR suite), the predictive error variance is minimized with respect to singular value number on each occasion that predictive error variance is calculated.

4.24.8 Error and Uncertainty Tables

A quick inspection of tables produced by GENLINPRED which record the outcomes of

corresponding types of error and uncertainty analysis performed on the same prediction reveal the following.

1. Predictive error variance is greater than predictive uncertainty variance.
2. Some parameter contributions to predictive error variance can be slightly negative.
3. The worth of some observations, as assessed through their ability to lower potential predictive error variance, can be slightly negative.

Unfortunately, analysis of predictive error is not as "clean" as that of predictive uncertainty. It is a "granular" procedure as it depends on a (necessarily discontinuous) number of singular values employed in the notional truncated SVD calibration exercise through which it is assessed. Furthermore, it is not a Bayesian procedure; error limits are less statistically efficient than are uncertainty limits. However where parameter and observation numbers are large, and hence where uncertainty analysis may become numerically difficult because of the size of the matrices which must be inverted in computation of predictive uncertainty, it provides a practical means of achieving such an analysis. However where error and uncertainty analysis can both be performed, uncertainty analysis provides a better indicator of post-calibration predictive variability, and statistics related thereto, than does error analysis; hence the outcomes of uncertainty analysis should be used in preference to the outcomes of error analysis. (Note that where observation numbers are large, consideration should be given to the use of a reduced number of super observations for implementation of PREDUNC-based uncertainty analysis; see the SUPOBSPREP utility.)

4.24.9 Identifiability

Computation of parameter identifiability requires that an estimate be provided of the dimensionality of the calibration solution and null spaces. SUPCALC provides such an estimate. However its estimate is often in error, with a distinct tendency for it to err on the side of too high a solution space dimensionality, and too small a null space dimensionality. The reason for this is that in most calibration contexts the bulk of "measurement noise" is in fact structural noise. Unfortunately this has a spatial and temporal correlation structure that SUPCALC (or anything else for that matter) cannot properly take into account. The fact that measurements show such unaccounted-for correlation actually diminishes their information content, and hence creates the existence of a null space of higher dimensionality than SUPCALC computes on the basis of zero or limited measurement noise correlation as implied in most real-world PEST control files. Hence there may be a need for the user to over-ride SUPCALC's estimate of the dimensionality of the calibration solution and null spaces with a smaller estimate of the former (and thereby a larger estimate of the latter).

4.24.10 Pre- and Post-Calibration Parameter Contributions to Error/Uncertainty

Based on its running of PREDVAR4 and PREDUNC4, GENLINPRED tabulates pre- and post-calibration contributions to predictive error/uncertainty by different parameter groups (or by individual parameters). Bar charts which depict these quantities are very informative, for they convey to the user (amongst other things):

1. the effectiveness (or otherwise) of the calibration process in reducing the error/uncertainty of critical predictions required of the model;
2. the parameter types that still contribute significantly to that error/uncertainty, even after the model has been calibrated.

Plots such as these will often reveal that the post-calibration contribution that a parameter makes to the error/uncertainty of a prediction of interest is greater than its pre-calibration contribution. This enigmatic occurrence is an outcome of the definition of "contribution to error/uncertainty" made by a particular parameter (or parameter group). This is defined as the decrease in error/uncertainty of the prediction of interest accrued through gaining perfect knowledge of the parameter in question (or of all parameters within a defined parameter group).

Prior to calibration, the uncertainty of a particular prediction may have no relation to that of a certain parameter (or group of parameters), because the prediction may be insensitive to that parameter (or group of parameters). However that prediction may be sensitive to one or more parameters with which the first parameter (or group of parameters) is correlated through the parameter estimation process (because they share one or more common null space eigencomponents). This means that perfect knowledge of the first parameter (group) allows better estimation of the second parameter (group) to take place through the model calibration process; hence acquisition of better knowledge of the first parameter (group) reduces the uncertainty of the prediction of interest, notwithstanding the fact that this prediction is insensitive to it (them). The post-calibration "contribution" made by members of the first parameter (group) to the uncertainty of the prediction of interest may therefore be significant, even though its pre-calibration contribution is small.

4.24.11 PREDUNC Uncertainty Formulation

As documented in descriptions of PREDUNC1, PREDUNC4 and PREDUNC5, these programs provide two different options for computation of predictive uncertainty and of quantities which depend on this. One of these options is better used where parameter numbers are small while the other is more efficient where observation numbers are small. GENLINPRED chooses the "efficient if low parameter numbers" option if the number of adjustable parameters in the PEST control file is less than the number of adjustable observations, and chooses the "efficient if low observation numbers" option otherwise. Nevertheless where both observation and parameter numbers are high, the run times associated with PREDUNC* programs, and hence with GENLINPRED, may be high.

4.24.12 Flexibility

As discussed above, GENLINPRED performs a variety of analyses, with the user having many choices over the analyses that are actually undertaken on any given run. Should further choices be required, the user should run the programs that GENLINPRED runs him/herself (for example, members of the PREDVAR and PREDUNC suites), in order to undertake the various types of analysis provided individually by each of the members of these suites.

4.25 Exploring Uncertainty using Pareto Concepts

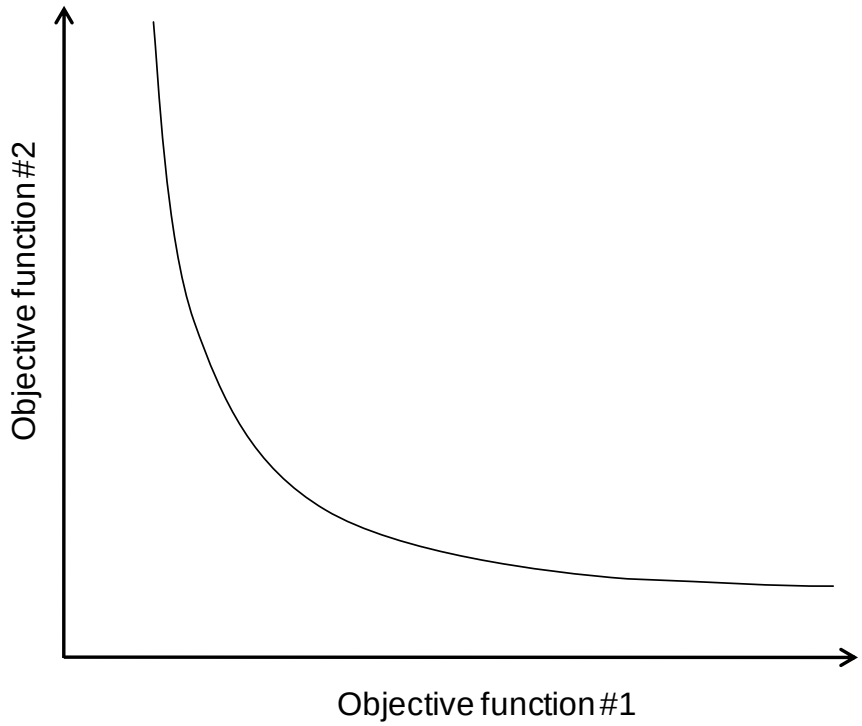
4.25.1 General

This section complements section 2.43 of this manual where the use of PEST's "Pareto" mode in a regularized inversion is discussed.

When an objective function is comprised of two or more components it is often impossible to minimize all of these simultaneously. Instead, one must be minimized at the cost of raising another. A modeller may then seek a tradeoff between different objective function components in finding an overall objective function at which the model is deemed to be

"calibrated". Alternatively he/she may wish to learn more about the nature of the tradeoff between the objective function components, thereby acquiring some insight into the information content of the data, and/or into model structural inadequacies.

PEST now provides functionality for exploring the nature of the tradeoff between two different objective functions. The hypersurface in parameter space which defines the tradeoff region between multiple objective functions is known in optimization parlance as the "Pareto front". Where the tradeoff is between only two objective function components, this front becomes a line. The figure below schematically depicts the Pareto front in a context marked by two objective function components.



Depiction of the Pareto front.

Mathematically, in the dual objective function context, the Pareto front can be characterized as the locus of points in parameter space for which it is impossible to improve both objective functions simultaneously. At one end of the Pareto front one of the two objective function components is minimized; at the other end of the Pareto front the other objective function component is minimized. In between these two endpoints, neither component is minimized; however neither can be lowered without raising the other. Hence no feasible objective function component combinations can exist to the left of the Pareto front.

The concept of the Pareto front can prove useful in calibration-constrained model predictive uncertainty analysis. In this case one objective function is comprised of members of the calibration dataset. In an under-determined calibration context this will need to include (extensive) prior information, similar to that used in Tikhonov regularization. This ensures that parameters remain realistic throughout the ensuing Pareto front optimization process; it also ensures uniqueness of that end of the front. The other objective function component can pertain to one or more appropriately-weighted "observations" comprising a prediction of future system behaviour. To the extent that the making of this prediction incurs model-to-measurement misfit, and/or the necessity for parameters to assume unrealistic values, the prediction becomes unlikely, this being measured by the increase in the calibration

component of the objective function, the latter rising as the prediction objective function component falls. Through exploration of the tradeoff between the two objective function components, as can be done through definition and traversal of the Pareto front, various predictions can be provided with formal or informal confidence levels. For more details of this process, including theory, see:

> Moore, C., Wöhling, T., and Doherty, J., 2009. Efficient regularization and uncertainty analysis using a global optimization methodology. *Water Resources Research* Under review at the time of writing.

In exploring the Pareto front, PEST starts at one end (the calibration end) and slowly moves towards the other. It is assumed that the model has already been calibrated, and that the objective function minimized through this calibration exercise is the same as that defining the calibration end of the Pareto front. (This will be referred to as "the calibration objective function" from now on.) Hence it is assumed that one end of the Pareto front has thereby been located. The other objective function component (henceforth referred to as the "prediction objective function" is then slowly introduced to the optimization process. Weights associated with observations which contribute to this objective function are zero at first, but then slowly increase at a user-specified rate. As PEST undertakes a sequence of optimization iterations in each of which the total objective function is minimized (as is its usual behaviour), the prediction objective function thus receives a greater and greater "hearing" in the overall objective function. Hence as the iteration count increases, PEST's location on the Pareto front changes from its original location at the calibration end as it moves towards the prediction end. When the weight associated with the prediction objective function rises to a user-specified level, the journey ends. It is often unnecessary for this journey to terminate at the prediction end of the Pareto front; however at least part of the front will have been traversed through this process. Hopefully enough of it will have been traversed for predictions that are encountered along the traversed part of the front to be assigned confidence levels, these being based on the degree to which the calibration objective function has been raised in lowering the predictive objective function, and hence in attaining respective predictions.

4.25.2 Implementation

To engage in exploration of the Pareto front PEST must be run in "Pareto" mode. For this to occur, the PESTMODE variable in the "control data" section of the PEST control file must be set to "Pareto". At the same time, a "Pareto" section must be present at the end of the PEST control file. In addition to this, initial parameter values in the PEST control file should correspond to the calibration end of the Pareto front; this is further discussed in the next subsection.

The figure below shows variables which appear in the "Pareto" section of the PEST control file. Following that is an example of this section.

```
* pareto
PARETO_OBSGROUP
PARETO_WTFAC_START PARETO_WTFAC_FIN NUM_WTFAC_INC
NUM_ITER_START  NUM_ITER_GEN NUM_ITER_FIN
ALT_TERM
OBS_TERM ABOVE_OR_BELOW OBS_THRESH NUM_ITER_THRESH
NOBS_REPORT
OBS_REPORT_1 OBS_REPORT_2 OBS_REPORT_3
```

Variables appearing in the "Pareto" section of the PEST control file.

```
* pareto
prediction
0.0  1.0  20
3  2  1
1
predmaxQ below 50 3
3
predmaxQ simflow2_3 simflow2_4
```

An example of the "Pareto" section of the PEST control file.

The first line of the "Pareto" section of the PEST control file must contain one entry, this being the name of an observation group cited in the PEST control file. This group must possess at least one observation of non-zero weight. PEST applies a changing multiplier to the weights associated with all observations belonging to this group as a means of exploring the Pareto front. The objective function component associated with this observation group thus becomes one of the two objective function components (the prediction objective function component) in the tradeoff which defines the Pareto front. The other objective function component involved in definition of the Pareto front (the calibration component) is the sum of all contributions made to the objective function by all other observation groups featured in the PEST control file.

In many instances of PEST-based Pareto front traversal the PARETO_OBSGROUP observation group will possess only one observation, this being a prediction whose probability of occurrence is being tested through the Pareto front traversal process.

The second line of the "Pareto" section of the PEST control file must contain two real numbers followed by an integer, these being respectively the PARETO_WTFAC_START, PARETO_WTFAC_FIN and NUM_WTFAC_INC variables. The first number is the initial weight factor that PEST should apply to members or the observation group PARETO_OBSGROUP (i.e. the predictive observation group). This defines the point at which PEST commences its journey along the Pareto front. In many contexts PARETO_WTFAC_START will be zero, this dictating that the journey commences at one extreme of the front, this being the calibration end of the front. PARETO_WTFAC is the final weight factor that PEST must apply to the PARETO_OBSGROUP observation group. In many cases this will be 1.0, for the user will have provided a weight (in the PEST control file) to the single member (or multiple members) of this group which will define the maximum amount that he/she would like the hypothesized prediction to find expression in the total objective function. (This of course is a matter of some subjectivity.) However there is no reason why PARETO_WTFAC_FIN cannot be greater than one.

Note that unless PARETO_WTFAC_FIN is provided with a very high value, it cannot be guaranteed that the prediction end of the Pareto front will be encountered in the Pareto journey undertaken by PEST. However, it is normally not necessary that this end of the Pareto front be located; indeed the prediction end of the Pareto front may not even correspond to a unique set of parameters, for there may be more than one way to abandon the constraints of the calibration dataset in achieving a particularly wayward prediction. All that is required in undertaking Pareto-based predictive uncertainty analysis is that the modeller define an appropriate "attractor prediction" or "observed prediction" that is somewhat different from the prediction made by the calibrated model, and assign a sufficiently high

weight to this prediction for enough of the Pareto front to be explored for likelihoods to be assigned to less extreme predictions.

The final variable on the second line of the "Pareto" section of the PEST control file is NUM_WTFAC_INC. This is the number of increments by which PEST should vary the weight factor in raising it from PARETO_WTFAC_START to PARETO_WTFAC_FIN. Thus, for example, if NUM_WTFAC_INC is supplied as 10, PEST will employ 11 (but possibly less - see below) weight factors in at least 10 (but possibly more) successive optimization iterations in exploring the Pareto front. The first weight factor will be PARETO_WTFAC_START, the second will be PARETO_WEIGHT_START plus a number equal to a tenth of the difference between PARETO_WTFAC_START and PARETO_WTFAC_FIN, etc.

The third line of the "Pareto" section of the PEST control file must contain three integers, these being NUM_ITER_START, NUM_ITER_GEN and NUM_ITER_FIN. These variables dictate how many optimization iterations PEST should devote to minimizing the total objective function (i.e. calibration plus prediction objective functions) based on each prediction weight factor that it employs in its journey along the Pareto front. NUM_ITER_START pertains to the first weight factor used (namely PARETO_WTFAC_START); NUM_ITER_START can be zero or greater. NUM_ITER_FIN pertains to the final predictive weight factor used (namely PARETO_WTFAC_FIN); this can also be zero or greater. NUM_ITER_GEN is employed for weight factors in between. This must be one or greater.

ALT_TERM, the sole variable on the fourth line of the "Pareto" section of the PEST control file, must be 0 or 1. If it is set to 1, then traversal of the Pareto front can terminate prior to using a weight factor of PARETO_WTFAC_FIN. In this case termination of PEST execution will occur if a user-nominated model output (i.e. an "observation" cited in the "observation data" section of the PEST control file) rises above, or falls below, a certain threshold, and stays above or below that threshold over a certain number of iterations. Specifics of this termination criterion must be supplied on the following line; this line must be absent if ALT_TERM is set to zero. The monitored observation is OBS_TERM (first variable on the next line of the "Pareto" section of the PEST control file). The next entry on this line must be "above" or "below". Then must follow the threshold itself, this being the variable OBS_THRESH. The number of successive iterations for which this threshold must be either exceeded or undercut is supplied as the ensuing NUM_ITER_THRESH variable; this must be 1 or greater. In many instances this observation will be the prediction whose likelihood is being examined through the Pareto front traversal process.

If NOBS_REPORT, on the following line of the "Pareto" section of the PEST control file is set to an integer greater than zero, then that number of model outputs (i.e. "observations" cited in the "observation data" of the PEST control file) are monitored and reported to the user as the Pareto front is traversed. If NOBS_REPORT is set to greater than 1, the names of these observations must be listed on the final line of the "Pareto" section of the PEST control file, these being the variables OBS_REPORT_1, OBS_REPORT_2 etc feature on that line.

4.25.3 Preparations for a PEST Pareto Run

It is the user's responsibility to ensure that initial values supplied in the PEST control file that is used as a basis for Pareto analysis correspond to that point on the Pareto front that is associated with the specified value of PARETO_WTFAC_START. In most cases PARETO_WTFAC_START will be specified as zero; hence initial values must correspond

to the calibration end point of the Pareto front. These parameter values can normally be found by undertaking a calibration exercise in which either the PARETO_OBSGROUP observation group is absent from the PEST control file, or in which all members of this group have been assigned weights of zero. (Recall that a new PEST control file can be built on the basis of optimized values from a previous PEST run using the PARREP utility.)

There is no reason why a "Pareto" mode PEST control file cannot contain prior information. If so, this will often be the same prior information that was employed in a previous Tikhonov calibration exercise. In fact, as stated above, in a highly parameterized context this prior information will assume the important role of maintaining parameter likelihood to as high degree as possible as an hypothesized prediction is tested through traversal of the Pareto front by preventing the occurrence of extreme parameter values. Designation of correct relative weighting between observations on the one hand and prior information equations on the other hand then becomes an important issue; weights on prior information equations must reflect the innate variability of hydraulic properties, while weights on observations must reflect the degree of measurement/structural noise associated with members of the calibration dataset. Collectively the prior information and observation components of the calibration dataset comprise the total calibration objective function that is traded off against the prediction in undertaking analysis of the uncertainty of this prediction.

Assignment of correct relative weighting to prior information on the one hand and field observations on the other will always be problematic. This is because the degree of innate variability that characterize system hydraulic properties on the one hand, and the degree of structural noise with which model-to-measurement misfit is contaminated on the other hand, can only be guessed. Nevertheless, in one form or another, such a decision must be made whenever calibration-constrained predictive uncertainty analysis is undertaken by this or any other means. Some strategies that may be followed in establishing credible relative weighting between prior information equations and observations to which model outputs must be matched include the following.

1. Work in terms of observation and parameter *differences* rather than in terms of *absolutes* when traversing the Pareto front. That is, let the parameters that are adjustable through the Pareto front exploration procedure be departures of parameters from their calibrated values, and let "observations" to which model outputs are matched be model outputs produced on the basis of the parameter set which calibrates the model. Optionally, project prior information into the calibration null space. Setup for this option can be facilitated through use of the PARREP, OBSPREP and REGPRED utilities. See the pertinent section of this manual for a description of these utilities, and of their philosophical basis.
2. When using PEST in a Tikhonov-based calibration exercise which precedes Pareto-based predictive uncertainty analysis, do not run PEST in "regularization" mode. Instead use fixed weights for Tikhonov prior information and observations, these weights reflecting innate hydraulic property variability and measurement/structural noise as discussed above.
3. In the Pareto predictive analysis process, use prior information weights that were calculated by PEST during a previous Tikhonov calibration exercise. Final weights obtained through the latter process can be obtained from the residuals file (i.e. the RES file) written by PEST upon conclusion of the regularized inversion process. (It may be important to set the "regcontinue" variable to "continue" during this calibration exercise).
4. Adjust regularization weights so that the contribution to the calibration objective

function made by observations and prior information equations are about the same at the commencement of the Pareto exploration process.

4.25.4 PEST Output Files

Some of the output files which PEST writes under normal circumstances are not written when PEST runs in Pareto mode; this is further discussed below. However two output files are written only when PEST runs in Pareto mode. These are the "Pareto objective function file", and the "Pareto parameter data file". The first of these is named ***case*.pod** whereas the second is named ***case*.ppd**, where ***case*** is the filename base of the PEST control file. The first is a text file, and can be inspected at any time during the course of the Pareto front traversal process. The second is a binary file; utilities are available for extracting some or all of the contents of this file and rewriting them in ASCII format.

The Pareto objective data file is easily exported into a plotting package, or into a spreadsheet, for plotting of the Pareto front (or of other information pertaining to this front). Data within this file is arranged in columns; a header is provided for each column to identify its contents. The file is updated at the end of every optimization iteration (including the "zeroth" iteration, this notionally comprising the initial model run based on parameter values supplied in the "parameter data" section of the PEST control file). The Pareto objective function file contains the contribution to the total objective function made by every observation group cited in the PEST control file, including that whose weight is varied as part of the Pareto front traversal process. It is important to note however, that the weight factor applied to this observation group in computing its contribution to the total objective function is 1.0. Hence weights applied to members of this group are full weights as supplied in the PEST control file. Thus by plotting values in this column against the sum of values in other objective function component columns, a picture of at least part of the Pareto front can be obtained (see the above graph).

In addition to objective function component columns, the Pareto objective function file contains a further NOBS_REPORT columns. These columns contain the values of those model outputs specified in the last line of the "Pareto" section of the PEST control file. If one of these is the sole member of the observation group whose weight is varied in the Pareto exploration process (this normally being the prediction whose uncertainty is explored), then a plot of this against the sum of all other objective function components provides a picture of how the value of the prediction must be traded off against calibration misfit. The rise in calibration objective function required to incur specific values of the prediction can be used to define predictive confidence intervals using formulas such as those presented in equations (2.10) and (2.11) of this document.

The Pareto parameter data file contains the same data as does the Pareto objective function file. However in addition to this it contains parameter values associated with each of the listed objective function values. This file can be converted to ASCII format using the PPD2ASC utility; note, however, that the width of this ASCII file will be large when many parameters are cited in the PEST control file. Alternatively, parameters listed in an individual record of the Pareto parameter data file can be extracted from that file using the PPD2PAR utility. Extracted parameters are written in parameter value file format. From there they can be inserted into a new PEST control file using the PARREP utility; a single model run can then be undertaken on the basis of these parameters by running PEST with NOPTMAX set to zero in this new PEST control file. PPD2ASC and PPD2PAR are discussed in more detail below.

In normal operation PEST writes a parameter value file at the end of each iteration of the parameter estimation process. This contains the "best" parameters achieved to date during the current PEST run. The meaning of "best" depends on the mode in which PEST is run. However when run in "Pareto" mode, the concept of "best" does not apply, for the process of optimization is replaced by one of Pareto front traversal. In this case PEST writes a new and distinct parameter value file at the end of each iteration. These files are named *case.par.0, case.par.1, case.par.2* etc, where *case* is the filename base of the PEST control file. The numeric extension associated with the name of each file pertains to the iteration number. An extension of "0" pertains to the end of the zeroth iteration, this being comprised of the initial model run undertaken on the basis of initial parameter values. Parameter values recorded in these files are the same as those recorded in binary form in the Pareto parameter data file (and in the ASCII-translated equivalent file written by PPD2ASC).

4.25.5 Some Differences and Similarities with Normal PEST Operation

When PEST runs in "Pareto" mode it informs the user through its screen output, and through its run record file, of the current value of the weight factor applied to the user-nominated PARETO_OBSGROUP observation group. This is done before the initial model run, and at the beginning of each optimization iteration. Objective function values written to the screen and recorded on the run record file for this observation group are calculated on the basis of this current weight factor (in contrast to objective function component values for this group recorded in the Pareto objective function and parameter data files which use full weighting as supplied in the PEST control file) This provides consistency with other aspects of iteration-specific PEST output, in particular the efficacy of different Marquardt lambdas in lowering of the objective function. Hence information written to the screen and recorded on the run record file is indicative of local optimization performance using the current prediction weight factor, but cannot be used for definition of the Pareto front.

Differences between normal PEST operation and its operation when run in "Pareto" mode arise from the fact that there is no actual "solution" to a Pareto front traversal process. Hence, as discussed above, multiple, iteration-specific, parameter value files are written in place of a single parameter value file containing "optimized" parameters. Another difference is that the residuals file (i.e. the RES file), and if appropriate the rotated residuals file (i.e. the RSR file), are not saved at the end of a PEST run. Nor is the intermediate residuals file (i.e. the REI file) saved at the end of every iteration. However if the REISAVEITN control variable is set to "reisaveitn", a suite of iteration-specific REI files are saved in the usual manner. The Jacobian matrix file (i.e. JCO file) is saved only at the end of the first optimization iteration. However iteration-specific JCO files are saved in the usual manner if the JCOSAVEITN control variable is set to "jcosaveitn".

Upon completion of execution, PEST does not run the model one last time using optimized parameter values (for there is no definition of "optimized" in the Pareto context). No resolution data file is saved, irrespective of the well- or ill-posedness of the optimisation problem with which PEST is faced during any optimization iteration, and of the solution mechanism chosen to solve a possibly ill-posed problem. When run in "Pareto" mode, PEST does not complete its run record file with a listing of optimized parameter values and a listing of parameter and observation statistics. Instead it simply announces that the Pareto front exploration process has reached completion.

With the exception of the NOPTMAX variable, termination criteria provided in the "control data" section of the PEST control file are ignored when PEST is run in "Pareto" mode, for

termination of traversal of the Pareto front is governed by variables residing in the "Pareto" section of the PEST control file. However if NOPTMAX is set to zero, then PEST will conduct only one model run before terminating execution; if it is set to -1 or -2 PEST will undertake enough model runs to calculate the Jacobian matrix, and then cease execution. In fact if NOPTMAX is provided with any of these values PEST will change its mode of operation to "estimation" so that its normal suite of outputs is recorded on its normal suite of output files before termination of execution.

If the FORCEN variable in the "parameter groups" section of the PEST control file is set to "switch" or "switch_5", PEST will cease execution with an error message. For the sake of consistency, the means by which finite-difference derivatives are calculated should not change as the Pareto front is traversed. (For maximum accuracy set FORCEN to "always_3", or even "always_5" if computing resources allow this.)

When run in "Pareto" mode, any of the usual methodologies offered by PEST for objective function improvement can be selected. Thus, for example, the PEST control file can contain a "singular value decomposition" or "LSQR" section; or it can specify that automatic user intervention enhance optimization performance. These mechanisms may indeed be necessary in parameter estimation contexts that are under-determined.

Other important similarities between "Pareto" mode PEST operation and normal PEST operation include the following.

1. Pareto front exploration can be undertaken using Parallel PEST or serial PEST.
2. PEST execution can be stopped at any time using the PSTOP or PSTOPST utilities.
3. If PEST execution is halted, it can be restarted using any of the normal restart switches.

4.25.6 Pareto Mode and PARREP

If, on completion of a Pareto run, the user decides that he/she needs to explore more or the Pareto front than was explored during the previous run, the following steps should be taken in order to commence a new PEST run as a continuation of a previous PEST Pareto run.

1. Using the PARREP utility, create a new PEST control file in which initial values are those computed at the end of the last iteration of the previous PEST Pareto run.
2. Set PARETO_WTFAC_START in the "Pareto" section of this new file to the same value as PARETO_WTFAC_FIN from the previous PEST run. (Unless NUM_ITER_FIN was zero for this previous run, in which case PARETO_WTFAC_START should be set to the last weight factor that PEST actually employed in its previous run.)
3. Select an appropriate value for PARETO_WTFAC_FIN.
4. Run PEST.

4.25.7 The PPD2ASC Utility

The PPD2ASC utility reads data that is stored by PEST in the binary Pareto parameter data file. This file contains the following information:

1. the contribution made to the objective function by each observation group, both at the commencement of the Pareto front exploration process, and at the end of every optimization iteration that PEST undertakes in exploring the Pareto front (weights applied to the adjustable group are full weights as in the objective function file);
2. the values of user-specified model outputs corresponding to these objective functions;

3. parameter values corresponding to the above.

The first two of the above items are also recorded by PEST in its ASCII Pareto objective function file; however storage of parameters is unique to the Pareto parameter data file. Binary storage is employed as this file would become unwieldy in highly parameterized contexts where parameters may number in the hundreds, or even in the thousands.

PPD2ASC stores all of the data recorded in the Pareto parameter data file in tabular ASCII format. It is run using the command:

```
ppd2asc ppdfile textfile
```

where:

ppdfile is the name of a PEST-generated Pareto parameter data file, and

textfile is the name of the ASCII file to which data housed in this file is transferred.

The text file written by PPD2ASC is readily imported into a spreadsheet for analysis or plotting.

4.25.8 The PPD2PAR Utility

Like PPD2ASC, PPD2PAR reads a binary Pareto parameter data file. It extracts a single parameter set from this file, recording this parameter set in parameter value file format. Once in this format, parameter values are available for use by a number of other PEST utilities. For example the PARREP utility can be used to write a PEST control file in which these parameter values are featured as initial values. If NOPTMAX is set to zero in this new PEST control file, PEST will run the model once, record objective function components, and then cease execution. Meanwhile model input files will contain these same parameter values, while model outputs will be calculated on the basis of these parameter values.

PPD2PAR is run using the command:

```
PPD2PAR ppdfile parfile N
```

where:

ppdfile is the name of a PEST-generated Pareto parameter data file,

parfile is the name of the parameter value file whose task it is for PPD2PAR to write, and

N is the iteration number pertaining to the extracted parameter set.

Parameter set *N* pertains to the end of the *N*'th iteration undertaken by PEST; an *N* value of zero corresponds to initial parameter values as provided in the PEST control file on which basis traversal of the Pareto front was undertaken.

Objective function values corresponding to any parameter set are available in ASCII form in the Pareto objective function file written by PEST. The parameter set corresponding to iteration *N* recorded in binary form in the Pareto parameter data file (this being the *N*+1th parameter set recorded in that file as counting begins at zero) corresponds to the *N*+1'th set of objective functions recorded in the Pareto objective function file. (The first objective function in that file corresponds to initial parameter values too.) Thus a point of interest on the Pareto front can be selected using a graph based on data from the Pareto objective function file; the parameter set corresponding to that point can then be extracted from the Pareto parameter data file using PPD2PAR.

The parameter set obtained by running PPD2PAR for a particular *N* corresponds to that residing in the parameter value file *case.par.N* written by PEST as it runs in Pareto mode.

4.26 PREDUNC7

The PREDUNC7 utility is similar to the PREDUNC1 utility. However instead of computing the uncertainty of a prediction, it computes the covariance matrix pertaining to the posterior parameter probability distribution. This can be computed in either of two ways.

$$\mathbf{C}'(\mathbf{p}) = [\mathbf{X}^t\mathbf{C}^{-1}(\boldsymbol{\varepsilon})\mathbf{X} + \mathbf{C}^{-1}(\mathbf{p})]^{-1} \quad (4.50)$$

or:

$$\mathbf{C}'(\mathbf{p}) = \mathbf{C}(\mathbf{p}) - \mathbf{C}(\mathbf{p})\mathbf{X}^t\left[\mathbf{X}\mathbf{C}(\mathbf{p})\mathbf{X}^t + \mathbf{C}(\boldsymbol{\varepsilon})\right]^{-1}\mathbf{X}\mathbf{C}(\mathbf{p}) \quad (4.51)$$

See documentation of PREDUNC1 for further details. Note that computation of C'(**p**) can take a long time where many parameters are featured in a PEST control file.

PREDUNC7's prompts are as follows:

```
Enter name of PEST control file:
Enter observation reference variance:

Enter name of parameter uncertainty file:
Enter name for posterior parameter covariance matrix file:

Use which version of linear predictive uncertainty equation:-
    if version optimized for small number of parameters   - enter 1
    if version optimized for small number of observations - enter 2
Enter your choice:
```

Equation (4.50) is the version of the equation which is "optimized for a small number of parameters" whereas equation (4.52) computes C'(**p**) in a manner that is numerically optimal where parameters outnumber observations. Where the parameter estimation problem is well posed, option (1) is the better choice.

The covariance matrix is written in PEST matrix file (ASCII) format to a file of the user's choice. Each row of the matrix is wrapped after each eight numbers.

In calculating C(**ε**) PEST divides weights provided in the "observation data" section of the PEST control file by the square root of the reference variance supplied in response to the second of the above prompts. If model-to-measurement misfit is entirely the result of measurement noise, and no account needs to be taken of structural noise, the reference variance can be approximated as the (measurement) objective function achieved through calibration divided by the sum of non-zero-weighted observations; do not include prior information in this equation.

5. Matrix Manipulation Programs

5.1 General

A series of matrix manipulation programs has been written to assist in carrying out uncertainty analysis in the post-processing of regularised PEST runs. They allow processing of a more general nature to be undertaken than that provided by the utilities described in the previous section. It is hoped that they thus provide the user with a certain degree of flexibility in accommodating the demands of their particular modelling applications.

All of these utilities read and write "matrix files" whose format follows that described in Section 4.4.3.

5.2 COV2COR

COV2COR calculates a correlation coefficient matrix from a covariance matrix. It reads the former from a file which observes matrix file format and writes the latter to a file in the same format. The matrix in the former file must satisfy the following criteria.

1. It must be square.
2. None of its diagonal elements must be zero or less.
3. Its row and column names must be identical. (The existence of non-identical row and column names is only possible if the ICODE value of the matrix file holding the first matrix is 2.)

COV2COR is run using the command:-

```
cov2cor covmatfile cormatfile
```

where:-

covmatfile is the name of the matrix file holding the covariance matrix, and

cormatfile is the name of the file to which the corresponding correlation coefficient matrix is written.

5.3 COVCOND

Suppose that an arbitrary vector $\mathbf{x}$ is partitioned into two separate parts $\mathbf{x}_1$ and $\mathbf{x}_2$. That is:-

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$$

Let C($\mathbf{x}$), the covariance matrix of $\mathbf{x}$, be correspondingly partitioned as:-

$$C(\mathbf{x}) = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix}$$

Suppose further that the elements of $\mathbf{x}_2$ become known. Then, if there is correlation between at least some members of $\mathbf{x}_2$ and some members of $\mathbf{x}_1$ (resulting in non-null $\mathbf{C}_{12}$ and $\mathbf{C}_{21}$ submatrices), the conditioned $\mathbf{C}_{11}$ matrix $\mathbf{C}'_{11}$ is calculable as:-

$$\mathbf{C}'_{11} = \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}^{-1}{}_{22}\mathbf{C}_{21}$$

COVCOND computes $\mathbf{C}'_{11}$, that is, the conditional covariance matrix of a subset of random variables, based on the premise that the complimentary subset becomes known. It is run using the command:-

```
covcond covfile1 listfile covfile2
```

where:-

covfile1 contains a covariance matrix (i.e. C(**x**) in the above explanation);

listfile contains a list of random variables whose values are assumed known, and

covfile2 contains the conditional covariance matrix based pertaining to the remaining variables (i.e. $\mathbf{C}'_{11}$ in the above explanation).

It will be recalled from elsewhere in this manual, that a PEST-compatible matrix file includes a list of row and column names. For a covariance matrix file row and column names will be the same; presumably, these will be the names of random variables whose stochastic structure is described by the covariance matrix. The listing file (this being the second COVCOND command line argument) must contain a list comprised of the names of some of these variables, written one to a line. No header, or other information, is required in this file.

5.4 JCO2MAT

JCO2MAT reads a PEST-produced Jacobian matrix file. It re-writes the matrix contained therein in the same matrix file format as that employed by other utility programs documented herein. The Jacobian matrix is then amenable to manipulation using these utilities.

JCO2MAT is run using the command:-

```
jco2mat jcofile matfile
```

where:-

jcofile is the name of a Jacobian matrix file, and

matfile is the name of the matrix file to which the Jacobian matrix is to be written.

5.5 JROW2MAT

JROW2MAT extracts a row of a Jacobian matrix from a PEST unformatted "Jacobian matrix file" and writes that row as a 1×NPAR matrix to a standard matrix file, where NPAR is the number of (adjustable) parameters featured in the Jacobian matrix file. Note that the expected extension for a Jacobian matrix file is "*.jco*".

JROW2MAT is run using the command:-

```
jrow2mat jcofile obsname matfile
```

where:-

jcofile is the name of a Jacobian matrix file,

obsname is the name of an observation or prior information item featured in that file, and

matfile is the name of the matrix file to which the 1×NPAR matrix is to be

written.

JROW2MAT simply extracts the nominated row of the Jacobian matrix and re-writes that row in appropriate format. The "row name" of the row is that of the extracted observation or prior information equation, while "column names" are the names of parameters featured in the Jacobian matrix file.

The reader is reminded that a PEST unformatted Jacobian matrix file can be viewed in ASCII format using the JACWRIT utility.

5.6 JROW2VEC

JROW2VEC performs the same function as JROW2MAT followed by MATTRANS. That is, it extracts a user-nominated row from a Jacobian matrix. However instead of writing the extracted row in the form of a row matrix, it writes it as a column matrix. Thus, if desired, the extracted set of observation or predictive sensitivities is immediately useable by MATQUAD for evaluation of predictive error variance.

JROW2VEC is run using the command:-

```
jrow2vec jcofile obsname matfile
```

where:-

jcofile is the name of a Jacobian matrix file,

obsname is the name of an observation or prior information item featured in that file, and

matfile is the name of the matrix file to which the NPAR×1 matrix is to be written.

5.7 MAT2JCO

MAT2JCO carries out the inverse of the operation carried out by JCO2MAT. It reads a matrix file and re-writes the matrix contained therein as a binary Jacobian matrix (i.e. JCO) file.

MAT2JCO can be useful for transporting Jacobian matrices between different computing platforms. For example a binary JCO file produced on a UNIX platform may not be readable by the version of PEST running on a PC because of the fact that the UNIX and PC versions of PEST will necessarily have been compiled using different compilers. To overcome this problem, the binary JCO file can be translated to ASCII matrix file format on the UNIX machine; the matrix file can then be transferred to the PC; re-translation to binary JCO format can then be effected using the MAT2JCO utility.

5.8 MAT2SRF

MAT2SRF rewrites a matrix in SURFER grid file format. This is a particularly useful device for viewing the resolution matrix. A highly diagonally dominant resolution matrix indicates (as the name suggests) good "resolution" of parameters through the parameter estimation process. A "blurry" resolution matrix which is only mildly diagonally dominant indicates inability of the calibration process to capture parameterisation detail, this arising from inadequacies in the information content of the calibration dataset as an outcome of data scarcity, data noise, or both.

MAT2SRF can read any matrix file which adheres to the format documented in Section 4.4.3 of this addendum.

MAT2SRF is run using the command:-

```
mat2srf matfile gridfile [threshold]
```

where:-

matfile	is the name of a matrix file;
gridfile	is the name of a SURFER grid file; and
threshold	is a blanking threshold.

Upon commencement of execution, MAT2SRF reads the matrix contained in the matrix file. It then rewrites the matrix contained therein in SURFER grid file format to file *gridfile*. Note that MAT2SRF automatically adds an extension of "*.grd*" to this filename unless it possesses this extension already.

MAT2SRF provides the option of blanking matrix elements whose absolute values are above a certain threshold. This threshold is optionally supplied as the last element of the MAT2SRF command line. If it is omitted, no such blanking takes place unless a matrix element has an absolute value greater than 1.70141e38, this being SURFERs "natural" blanking threshold.

When a matrix is plotted (and shaded/contoured) in SURFER, rows and columns of this matrix appear in the same order in the SURFER plot as they do in the numerical representation of the matrix in the corresponding matrix file.

5.9 MATADD

MATADD adds one matrix to another. It is run using the command:-

```
mattadd matfile1 matfile2 matoutfile
```

where *matfile1* and *matfile2* are files containing the matrices to be added, while *matoutfile* contains the file to which MATADD writes the summation of the two supplied matrices.

The following should be noted:-

1. Two supplied matrices must have the same number of rows and columns if they are to be added.
2. If row and column names differ in substance or in order between the two supplied matrix files, MATADD will transfer those provided in *matfile1* to the summation matrix recorded in file *matoutfile*. However it will warn the user of the name-incompatibility existing between the matrices contained in *matfile1* and *matfile2*. Note that MATADD will NOT re-order the rows and columns of one matrix in order to ensure correspondence with that of another matrix. If the number of rows and columns are the same in each matrix file, the matrices are simply added, and a warning message is written to the screen.

5.10 MATCOLEX

MATCOLEX stands for MATrix COLumns EXtract. Using this utility, the first *ncol* columns are extracted from a matrix and rewritten as a new matrix to a new matrix file. MATCOLEX is run using the command:-

```
MATCOLEX matfile ncol matoutfile
```

where *matfile* contains an arbitrary rectangular *m*×*n* matrix. A new *m*×*ncol* matrix is written to the matrix file *matoutfile*.

If *ncol* is supplied as negative, then the last *ncol* columns of a matrix are extracted; these columns are written to the new matrix in reverse order.

5.11 MATDIAG

MATDIAG extracts the diagonal of a matrix, writing it as a vector (i.e. a one-column matrix). However certain conditions must be met for MATDIAG to do its job. These are:

1. the matrix must be square;
2. the row and column names of the matrix must be the same.

MATDIAG is run using the command:-

```
MATDIAG matfile1 matfile2
```

where *matfile1* is the name of the file holding the diagonal matrix, and *matfile2* is the name of the matrix file to which the extracted diagonal elements are written. The rows of this one-column matrix are given the same names as the rows of original matrix. Its single column is given the name "col1".

5.12 MATDIFF

MATDIFF is identical to MATADD except for the fact that it undertakes matrix differencing rather than matrix addition.

5.13 MATINVP

MATINVP finds the inverse of a positive definite matrix. It is run using the command:-

```
matinvp matfile1 matfile2
```

where *matfile1* contains the matrix to be inverted and *matfile2* contains the matrix to which the inverse is written. If the matrix contained in *matfile1* is not positive definite, then MATINVP ceases execution with an appropriate error message.

5.14 MATJOINC

MATJOINC reads two matrices which have the same number of columns, and for which corresponding column names are the same in each matrix. It forms a new matrix, by joining these two matrices in the column direction. Thus suppose that the two existing matrices are named **A** and **B**. MATJOINC combines these matrices into a single matrix **C** of the form:-

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix}$$

MATJOINC is run using the command:-

```
matjoinc matfile1 matfile2 matfile3
```

where:-

matfile1 is the name of a matrix file holding the first matrix,

matfile2 is the name of a matrix file holding the second matrix, and

matfile3 is the new matrix file.

The following should be noted:-

1. MATJOINC will not join two existing matrices if any column name in the first matrix differs from that of its corresponding column in the second matrix.
2. MATJOINC will not join two existing matrices if any row name in the first matrix is the same as a row name in the second matrix (for then the resulting matrix would have duplicate row names).
3. Row and column names from the existing matrices are transferred to the new matrix.
4. If the resulting, combined, matrix is a diagonal matrix, MATJOINC writes the joined matrix in diagonal matrix format for the sake of storage efficiencies.

5.15 MATJOIND

MATJOIND reads two matrices of arbitrary dimensions. It then forms a new matrix in which the two are combined in a diagonal sense. Thus suppose that the two existing matrices are named **A** and **B**. MATJOIND combines these matrices into a single matrix **C** of the form:-

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}$$

MATJOIND is run using the command:-

```
matjoind matfile1 matfile2 matfile3
```

where:-

matfile1 is the name of a matrix file holding the first matrix,

matfile2 is the name of a matrix file holding the second matrix, and

matfile3 is the new matrix file.

The following should be noted:-

1. MATJOIND will not join two existing matrices if any column name in the first matrix is the same as a column name in the second matrix.
2. MATJOIND will not join two existing matrices if any row name in the first matrix is the same as a row name in the second matrix.
3. Row and column names from the existing matrices are transferred to the new matrix.
4. If the resulting, combined, matrix is a diagonal matrix (which can only occur if the two existing matrices are themselves diagonal), MATJOIND writes the joined matrix in diagonal matrix format for the sake of storage efficiencies.

5.16 MATJOINR

MATJOINR reads two matrices which have the same number of rows, and for which corresponding row names are the same in each matrix. It forms a new matrix, by joining these two matrices in the row direction. Thus suppose that the two existing matrices are named **A** and **B**. MATJOINR combines these matrices into a single matrix **C** of the form:-

$$\mathbf{C} = [\mathbf{A} \quad \mathbf{B}]$$

MATJOINR is run using the command:-

```
matjoinr matfile1 matfile2 matfile3
```

where:-

matfile1 is the name of a matrix file holding the first matrix,

matfile2 is the name of a matrix file holding the second matrix, and

matfile3 is the new matrix file.

The following should be noted:-

1. MATJOINR will not join two existing matrices if any row name in the first matrix differs from that of its corresponding row in the second matrix.
2. MATJOINR will not join two existing matrices if any column name in the first matrix is the same as a column name in the second matrix (for then the resulting matrix would have duplicate column names).
3. Row and column names from the existing matrices are transferred to the new matrix.
4. If the resulting, combined, matrix is a diagonal matrix, MATJOINR writes the joined matrix in diagonal matrix format for the sake of storage efficiencies.

5.17 MATORDER

The purpose of MATORDER is to re-order the rows and columns of a matrix. It is run using the command:-

```
MATORDER matfile1 matfile2 matoutfile
```

where:-

matfile1 is the name of a matrix file holding the matrix to be re-ordered,

matfile2 is the name of the "reordering matrix", and

matfile3 is the name of a new matrix file which will hold the reordered matrix.

MATORDER commences execution by reading the matrix contained in *matfile1*. It then reads *matfile2*. If the number of rows and columns in the *matfile2* matrix is not the same as the number of rows and columns in the *matfile1* matrix, MATORDER ceases execution with an appropriate error message. MATORDER then verifies that there are no duplicate column names in each of the *matfile1* and *matfile2* matrices, and that there are also no duplicate row names.

Next MATORDER reorders the *matfile1* matrix such that its rows have the same order as the rows of the *matfile2* matrix, and such that its columns have the same order as the columns of the *matfile2* matrix. It then records the re-ordered *matfile1* matrix in file *matoutfile*.

It is important to note that the matrix in file *matfile2* is not actually used by MATORDER; only the row and column names are used. Thus any matrix can be employed. If the matrix has a large number of rows and/or columns, and if file *matfile2* is being prepared by hand, don't forget the shorthand manner in which numbers can be read using the free field formatting convention. For example each row of a 1000-column nul matrix **0** can be represented using

the format "1000*0.0".

5.18 MATPROD

MATPROD calculates the product of two matrices. That is, it calculates **C** where **C**=**AB**. Note that calculation of **C** is only possible if the number of columns in **A** is equal to the number of rows of **B**. Ideally the names of the rows of **A** should be the same as the names of columns of **B**. MATPROD will not object if this is not the case; however it will issue a warning.

MATPROD is run using the command:-

```
MATPROD matfile1 matfile2 matoutfile
```

where *matfile1* and *matfile2* contain the **A** and **B** matrices respectively; *matoutfile* will contain the **C** matrix upon completion of MATPROD execution.

5.19 MATQUAD

MATQUAD evaluates the quadratic form $\mathbf{y}^t\mathbf{M}\mathbf{y}$ where **y** is a vector and **M** is a square matrix. It is run using the command:-

```
matquad vecfile matfile outfile
```

where:-

vecfile	is the name of a matrix file holding the vector **y**,
matfile	is the name of a matrix file holding the matrix **M**, and
outfile	is an output matrix file.

The following should be noted:-

1. MATQUAD requires an input vector **y**. However this vector is actually read as a $n\times1$ matrix from a standard matrix file.
2. Even though $\mathbf{y}^t\mathbf{M}\mathbf{y}$ is a scalar, MATQUAD writes this scalar as a 1×1 matrix to its output matrix file *outfile*. However it also writes it to the screen.
3. MATQUAD will issue a warning message if the names of the rows of the vector **y** are not the same as those of the rows of **M**. It will also issue a warning message if the rows of **M** are named differently to the columns of **M**.

5.20 MATROW

MATROW extracts a row of a matrix. It then re-writes that row as a "row matrix" to a matrix file.

An interesting usage of MATROW is the extraction of a row of the resolution matrix. MATTRANS can then be used to write row entires in the vertical direction rather than the horizontal direction. For those parameters which correspond to pilot points (or other point-based geographical entities) geographical coordinates are easily pasted in adjacent columns. Griding and contouring of this data then allows graphical viewing of the averaging process that attends the estimation of spatial model parameters.

MATROW is run using the command:-

```
matrow matfile rowname matoutfile
```

where:-

matfile is the name of a matrix file,

rowname is the name of a row of the matrix contained in *matfile*, and

matoutfile is the name of a new matrix file containing the nominated row of the first matrix.

5.21 MATSMUL

MATSMUL multiplies a matrix by a scalar. It is run using the command:-

```
matsmul matinfile number matoutfile
```

where:-

matinfile is a file containing a matrix,

number is the matrix scalar multiplier, and

matoutfile is the file to which the new matrix will be written.

5.22 MATSPEC

MATSPEC lists some useful matrix specifications to a nominated text file. As time goes on the matrix characteristics that are recorded to this file will expand. However at the time of writing, only the following information is recorded:-

1. number of rows and columns;
2. row/column numbers/names of highest and lowest matrix elements;
3. row/column numbers/names of highest and lowest absolute matrix elements;
4. row/column numbers/names of highest and lowest diagainal elements;
5. row/column numbers/names of highest and lowest absolute diagonal elements.

MATSPEC is run using the command:-

```
matspec matfile outfile
```

where *matfile* is the name of a file holding a matrix, and *outfile* is the name of the text file to which matrix properties are written.

5.23 MATSVD

MATSVD undertakes singular value decomposition of an arbitrary $m \times n$ square matrix. Suppose that this matrix is named **A**. Then singular value decomposition of **A** leads to computation of the matrices **U**, **S** and **V** where:-

$$\mathbf{A} = \mathbf{USV}^t$$

In the above equation **U** is an $m \times m$ orthogonal matrix, **V** is an $n \times n$ orthogonal matrix and **S** is a "rectangular diagonal" matrix of dimension $m \times n$ containing the singular values of **A**. These are real and non-negative, and are returned in descending order by MATSVD. The first min(m,n) columns of **U** and **V** are the normalised left and right singular vectors of **A**.

MATSVD is run using the command:-

```
matsvd matfile umatfile smatfile vtmatfile
```

where:-

matfile is a user-supplied matrix file containing an arbitrary rectangular matrix **A**,

umatfile contains the SVD-generated **U** matrix,

smatfile contains a square $j \times j$ **S** (singular value) matrix where j is the least of m and n, and

vtmatfile contains the SVD-generated $n \times n$ $\mathbf{V}^t$ matrix, where $\mathbf{V}^t$ is the transpose of **V**.

Note that while **S** as represented in the above equation is an $m \times n$ matrix, it is only recorded as a $j \times j$ matrix by MATSVD (singular values beyond this are zero). The fact that it is square allows it to be written with an ICODE value of zero. This, in turn, allows the user to much more easily inspect the singular values of **A** than if they were recorded in a large rectangular matrix of predominantly zero elements.

5.24 MATSYM

MATSYM reads a square matrix **A**. It forms a symmetric matrix as $(\mathbf{A} + \mathbf{A}^t)/2$, writing this matrix to a user-nominated file. MATSYM is run using the command:-

```
matsym      matfile matoutfile
```

where:-

matfile is the name of a matrix file containing a square matrix, and

matoutfile is the name of a new matrix file to which MATSYM will write the symmetric matrix, calculated as above.

5.25 MATTRANS

MATTRANS reads a matrix file. It writes another matrix file containing the transpose of the first matrix file. It is run using the command:-

```
mattrans matfile1 matfile2
```

where:-

matfile1 is the name of a matrix file, and

matfile2 is the name of a new matrix file containing the transpose of the first matrix file.

5.26 MATXTXI

MATXTXI calculates $(\mathbf{X}^t\mathbf{X})^{-1}$ where **X** is a user-supplied matrix for which the number of columns does not exceed the number of rows. It is run using the command:-

```
matxtxi matfile1 matfile2
```

where *matfile1* contains the **X** matrix. After completion of execution *matfile2* contains the matrix $(\mathbf{X}^t\mathbf{X})^{-1}$.

5.27 MATXTXIX

MATXTXIX calculates $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$ where **X** is a user-supplied matrix for which the number of columns does not exceed the number of rows. It is run using the command:-

```
matxtxi matfile1 matfile2
```

where *matfile1* contains the **X** matrix. After completion of execution *matfile2* contains the matrix $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$.

5.28 PEST2VEC

PEST2VEC reads a PEST control file. A template file of a single column matrix file is then written based on adjustable parameters cited within the PEST control file; so too is the matrix file itself, using initial parameter values recorded in the PEST control file. A complementary transformation vector file (second matrix file required by VECLOG) is also written. PEST2VEC also writes a new PEST control file in which the template file and associated matrix file are added to its "model input/output "section.

PEST2VEC assists in the process of allowing parameter value matrix manipulation to be implemented as part of the model run undertaken by PEST during the parameter estimation or predictive analysis (more likely the latter) process. The model, as run by PEST, will need to be upgraded (normally by adding lines to the model batch file) such that one or more of the matrix manipulation utilities documented herein are cited; if any parameters are log transformed the command to run VECLOG will need to precede all other matrix operations in this file. The result of such parameter manipulation is likely to be a matrix itself. VEC2PEST can be used to alter a PEST control file to accommodate the inclusion of these results in the parameter estimation and/or predictive analysis processes; VEC2PEST will also write an instruction file through which the outcomes of such matrix manipulation can be read by PEST.

PEST2VEC is run using the command:

```
PEST2VEC pestfile1 pestfile2 tplfile matfile logfile
```

Where:-

pestfile1	is an existing PEST control file,
pestfile2	is a new PEST control file written by PEST2VEC in which the template file and new matrix file are cited in the "model input/output" section,
tplfile	is a template file of a matrix file,
matfile	is a matrix file citing the initial values of adjustable parameters as recorded in the PEST control file, and
logfile	is the name of a transformation vector file for the use of VECLOG.

5.29 VEC2PEST

VEC2PEST is designed to facilitate the use of matrix manipulation outcomes as observations to be used by PEST in the parameter estimation and predictive analysis processes. This is expected to be particularly useful where parameters, or parameter projections, must suffer the constraints imposed by a covariance matrix when maximizing or minimizing a key model prediction as part of the predictive analysis process. Thus, as part of this process, PEST parameters can be written to a matrix file, logarithmically transformed as appropriate, and then possibly projected onto a subspace of parameter space using matrix multiplication utilities described herein. The outcome of such a parameter manipulation process will be a

single column matrix (i.e. vector) residing in a matrix file. VEC2PEST generates an instruction file using which the components of this vector can be read by PEST, and alters an existing PEST control file to include these vector elements as observations.

In undertaking these activities, VEC2PEST observes the following protocols. These may not be suitable for all occasions. Hence it is essential that the VEC2PEST-modified PEST input dataset be checked with the comprehensive error checking utility PESTCHEK.

1. Observation names are denoted as the names of the rows of the single column matrix to which they correspond.
2. An observation group name is formulated as a text string that is common to all of the new observation names; if such a string cannot be found, the name of the new observation group is provided as *mat_data*.

If these protocols result in conflicts with existing names, the situation must be remedied by direct editing of the PEST control and instruction files generated by VEC2PEST.

VEC2PEST is run using the command:

```
vec2pest vecfile pestfile1 pestfile2 insfile2 [covfile]
```

where:-

vecfile	is the name of a matrix file containing a single-column matrix;
pestfile1	is the name of an existing PEST control file to be modified by VEC2PEST;
pestfile2	is the name of a new PEST control file to be written by VEC2PEST containing the new observations;
insfile2	is the name of an instruction file through which the nominated matrix file will be read by PEST; and
covfile	is the name of an optional covariance matrix file for the new observation group.

In many instances, the covariance matrix file will have been prepared by the utility software documented herein, or by the PARAMERR utility. Note that VEC2PEST does not check for the existence of the optionally nominated covariance matrix file. It simply adds its name to the appropriate place of the PEST control file which it generates, opposite the name of the new observation group in the "observation groups" section of this file.

5.30 VECLOG

VECLOG reads two matrix files, each of which must contain a single column matrix (i.e. a vector). The second of these vectors must contain elements which are either 0, 1 or -1. For each element of the second vector which is 1, the corresponding element of the first vector is log (to base 10) transformed by VECLOG in forming the corresponding element of a new vector. For each element of the second vector which is -1, the corresponding element of the first vector becomes a power of 10 in computing the corresponding element of the new vector; that is, the corresponding element of the new vetor is computed as 10^x where x is the pertinent element of the first vector. For each element of the second vector which is 0, the corresponding element of the first vector remains unchanged, and is thus directly transferred to the new vector. After transformation in this manner, VECLOG records the new vector in a new matrix file.

VECLOG is run using the command:-

```
veclog matfile1 matfile2 matfile3
```

where:-

matfile1	contains the first vector,
matfile2	contains the second vector, and
matfile3	is the name of a new matrix file to which the log-transformed vector is written.

Note that if any element of the first vector for which log transformation is sought is zero or negative, VECLOG will cease execution with an appropriate error message. Note also that if any element of the new vector is computed to exceed 10^{36}, it is recorded as 10^{36} to avoid numerical overflow.

6. A Global Optimiser – CMAES_P

6.1 General

CMAES_P is a "PEST compatible" implementation of the powerful and robust CMA-ES global optimisation scheme. For details of this scheme see, for example, Hansen and Ostermeier (2001) and Hansen et al (2003). See Bayer and Finkel (2007) for an application of the algorithm to groundwater contaminant capture. See also Nikolaus Hansen's web pages at

http://www.bionik.tu-berlin.de/user/niko

for further details, including a tutorial as well as MATLAB and OCTAVE source code. Although a brief outline of the algorithm will be presented below, it is strongly suggested that the user learn something about this algorithm from these sources, so that he/she is equipped with the knowledge necessary to adjust various control variables in order to meet the demands of different problems.

Unlike PEST, CMAES_P does not require derivatives of model outputs with respect to adjustable parameters for implementation of its optimisation algorithm. Thus it can be employed where model outputs show "numerical granularity" due to model instability, or where the model is highly nonlinear and/or the objective function surface shows local minima at various scales. Where model derivatives have integrity, PEST's performance will almost certainly be superior to that of CMAES_P (in terms of the number of model runs required to find the objective function minimum). Where model derivatives do not have integrity, however, CMAES_P's performance may be superior to that of PEST.

CMAES_P was built from version 2.5 of CMA-ES. I would like to point out, however, that any problems, imperfections and inefficiencies encountered in using CMAES_P were probably introduced by myself rather than being characteristics of the algorithm. I apologize if I have inadvertently done this.

6.2 The Algorithm in Brief

"CMA" stands for "covariance matrix adaptation". "ES" stands for evolutionary strategy. The algorithm employed by CMAES_P is sometimes given the longer name "CMA-ES with rank μ update and weighted recombination" or simply "(μ, λ) CMA-ES".

The CMA-ES algorithm employs an iterative procedure to minimize an objective function which is dependent on n adjustable parameters. In each iteration of this procedure λ random realisations of n-dimensional parameter vectors are generated, and the objective function is computed for each (this requiring one model run in each case). Normally λ (also referred to as the "population size") is considerably smaller than n. The n-dimensional covariance matrix used to generate these parameter realisations evolves through the optimisation process in response to what is learned about the objective function surface from sampling it in this way. This maintains efficiency of the process, for as the minimum of the objective function is approached, the search radius about that minimum for possible lower objective functions is reduced in order to exclude from the n dimensional search area portions of parameter space in which it is extremely unlikely that the global minimum lies. On the other hand, the fact that the search process is random reduces the chances of being trapped in a local minimum of the objective function.

Once the λ model runs have been carried out, the parameter sets giving rise to the μ lowest

objective functions are selected, and a new "mean" parameter set calculated through formulating a weighted average of the corresponding μ parameter value sets (μ is sometimes referred to as the "number of parents"). Weights can be assigned as "super-linear", "linear", or "equal". In the first two cases, greater weight is given to parameters that give rise to lower objective functions, this often leading to faster reduction of that function. As will be discussed below, the CMAES_P default is "super linear" (in which lower objective functions are weighted more heavily than in the "linear" option); though in many cases "linear" works at least as well.

Once a new average set of parameter values has been computed in this fashion, the next iteration begins. Because random parameter realisations are generated to be symmetrical about this mean, there is a tendency of the objective function to fall as iterations proceed.

An extremely important component of the CMA-ES methodology, however, is its capacity to adapt the parameter covariance matrix that forms the basis for generation of random parameter samples as the optimisation process progresses. Adaptation takes place on the basis of wisdom gained from both the current iteration, and from previous iterations, the mix between the two being set internally by the CMA-ES algorithm. See the references cited above for further details.

The CMA-ES algorithm goes to considerable lengths in its handling of parameter bounds. Bounds are never violated, for if any member of a random parameter set violates its individual bound, it is adjusted to respect it. In addition to this the tendency to generate parameters which violate bounds is mitigated through adding a penalty to the objective function (internally to the code) as this occurs; this in turn affects the evolving parameter covariance matrix on the basis of which further parameter realisations are generated.

6.3 The CMAES_P Implementation of CMA-ES

6.3.1 Introduction

CMAES_P reads a PEST input dataset, including the PEST control file and template and instruction files cited therein. Like PEST, it communicates with a model through that model's own input and output files. Like PEST, it runs a model many times, maximizing the fit between model outputs and field measurements recorded in the PEST control file (unless it is asked to minimize a single model output which is assumed to represent a model-calculated objective function – see below).

6.3.2 Features in Common with PEST

CMAES_P has the following features in common with PEST. The reader is referred to the PEST manual for a further description of these features. In some case, however, a brief description is provided below.

1. The PEST input dataset read by CMAES_P can include prior information, multiple observation groups, and can use observation covariance matrices instead of observation weights. The objective function calculated by CMAES_P is exactly the same as that calculated by PEST.
2. Model runs can be undertaken in serial, or in parallel. The parallel run protocol is the same as that of PEST. If parallelisation is implemented, CMAES_P must be provided with a run management file. This has the same filename base as the PEST control file, but possesses an extension of "*.rmf*". The history of communication between

CMAES_P and its slaves is recorded in a run management record file. This has the same filename base as the PEST control file, but possesses an extension of "*.rmr*".

3. Like PEST, CMAES_P records the progress of the optimisation process both to the screen and to a run record file. The latter has the same filename base as the PEST control file, but possesses an extension of "*.rec*".
4. At any stage of the optimisation process, and at the end of this process, best parameters computed to date can be found in a "parameter value file", this having the same filename base as the PEST control file, but an extension of "*.par*".
5. A CMAES_P run can be terminated by issuing the PSTOP or PSTOPST command in another window. Its execution can be paused and unpaused using the PPAUSE and PUNPAUSE commands.

6.3.3 Parameter Estimation and Optimisation

PEST minimizes an objective function computed as the sum of squared weighted residuals between model outputs and field or laboratory measurements. So too does CMAES_P. However use of the CMA-ES algorithm is not restricted to an objective function computed in this manner, for it can be employed to minimize an objective function calculated in any way whatsoever (even an objective function whose minimum value is less than zero). While code internal to CMAES_P computes an objective function in the same way that PEST does, CMAES_P provides the option of minimizing a single model-generated number (this presumably being a model-calculated objective function). This option becomes available when the PEST control file read by CMAES_P contains a single observation and no prior information. In this case CMAES_P asks the user whether the single model output corresponding to the observation is to be treated as an objective function to be minimized, or whether it should be matched to the corresponding measurement in the PEST control file, thereby formulating a one-term least squares objective function to be minimized. In the former case, CMAES_P does not calculate an objective function internally, turning its attention instead to minimizing the single model output presented to it.

6.4 Preparing for a CMAES_P Run

6.4.1 The Input Dataset

As stated above, the input dataset for CMAES_P is identical to that of PEST. Hence utility software employed for automatic construction of a PEST input dataset can also be employed for construction of a CMAES_P input dataset. Certain features of this dataset, however, are ignored (for example variables which govern the calculation of finite–difference derivatives, for the CMA-ES algorithm does not employ these derivatives). Control variables required by the CMA-ES algorithm are solicited from the user upon commencement of CMAES_P execution.

The CMAES_P input dataset can include most of the features available through a PEST input dataset, including the following.

1. Not all parameters cited in the PEST control file need be adjustable, for some parameters can be fixed or tied. In the latter case, the parent parameter to one or more tied parameters is adjusted while the tied parameters simply "ride on its back".
2. Parameters can be log-transformed or untransformed. In the former case the CMA-ES algorithm actually estimates the logs of pertinent parameters. The same rationale that

underpins selection of the log transformation option when using PEST should underpin its selection when using CMAES_P.

3. Parameters can assigned a non-unity SCALE and/or a non-zero OFFSET.
4. Prior information can be included in the PEST control file.
5. The model run by CMAES_P can be a single executable, or a batch/script file comprised of many executables.
6. A covariance matrix can be employed instead of weights for user-specified groups of observations.

Nevertheless, there are some restrictions on a CMAES_P input dataset which do not apply to a PEST input dataset. These include the following.

1. The PESTMODE variable must be set to "estimation". It must not be set to "prediction" or "regularisation". See the discussion below, however, for a way in which CMAES_P can undertake "pseudo-regularisation".
2. If the NUMCOM variable in the "control data" section of a PEST control file is set to greater than 1, and therefore multiple commands appear in the "model command line" section of the PEST control file. CMAES_P uses only the first of these to run the model.
3. A non-zero setting of MESSFILE is ignored. Thus CMAES_P does not write a PEST-to-model-message file.
4. A non-zero setting of JACFILE is ignored, as are the contents of the "derivatives command line" section of a PEST control file.

6.4.2 Parameter Transformation

As stated above, if parameters are designated as log-transformed during the optimisation process, the CMA-ES algorithm only "sees" the log of these parameters. Thus the internal parameter covariance matrix used in the generation of samples of parameter space will in fact pertain to the logs of pertinent parameters.

As is discussed in the PEST manual, many problems are more linear (and hence more easily solved) if formulated in terms of the logs of parameters, rather than in terms of native parameters. Log transformation also has the ability to make parameter sensitivities more equal, and hence reduce elongation of the parameter covariance matrix. However use of an appropriate parameter SCALE can have the same affect, and can be more appropriate if log transformation introduces a significant amount of nonlinearity. Note that a negative SCALE and/or an appropriate OFFSET can maintain positivity in a parameter prior to its log-transformation if the latter is deemed to be useful in certain optimisation contexts.

6.4.3 Parameter Initial Values and Parameter Bounds

As is documented in the PEST manual, upper and lower bounds can be placed on parameter values in the "parameter data" section of the PEST control file; these will be respected as the optimisation process progresses.

In normal PEST usage, parameter bounds serve a number of roles. On the one hand they can provide a "reality check" on parameters estimated through the optimisation process. On the other hand they can prevent model instability or other errors that attend the transgression of suitable parameter limits.

When using CMAES_P, bounds serve another important role, of which the user should be aware. *Initial parameter standard deviations are computed as one fourth of the interval between user-supplied upper and lower bounds.* As a result of this, random parameter realisations generated at the start of the optimisation process potentially occupy the entirety of the bounded interval. However there will be a tendency for these to be closer to initial parameter values than to parameter bounds, as the former are provided to the CMA-ES algorithm as initial parameter mean values, and hence values of highest likelihood from the random parameter generation point of view.

In recognition of the important role played by bounds and initial values, the following rules should be followed if possible.

1. Bounds should NOT be placed at "plus and minus infinity" (using very high and low numbers) as is sometimes done when using PEST.
2. If appropriate, bounds should be symmetrical about parameter initial values. If parameters are log-transformed, then the logarithms of parameter bounds should be symmetrical about the logarithms of initial parameter values.

It is important to note, however, that initial parameter standard deviations computed in this manner can be overridden through supplying an initial parameter covariance matrix – a matter that is discussed in more detail below.

6.5 Running CMAES_P

6.5.1 The CMAES_P Command Line

To inspect CMAES_P command-line options, simply type its name at the screen prompt. CMAES_P will respond by writing the following text to the screen.

```
CMAES_P is run using the command:

    cmaes_p pestfile [/p] [/r]

where

    pestfile is the name of a PEST control file,
    /p       is an optional parallelisation switch, and
    /r       is an optional restart switch.
```

As was discussed in a previous section, the input dataset for CMAES_P is identical to that of PEST, including a run management file if model runs are to be parallelised. However the serial and parallel versions of CMAES_P employ the same executable. This is further discussed below.

As the optimisation process progresses, CMAES_P records the current value of the objective function to the screen. If model runs are undertaken in serial, then CMAES_P screen output is interspersed with that of the model as both CMAES_P and the model share the same window. In this case it may be convenient to redirect model output to a "nul file".

6.5.2 CMAES_P Prompts

After having read the PEST input dataset, CMAES_P presents the user with a series of prompts. But first, if it has established that:-

1. there is only one observation cited in the PEST control file, and
2. there is no prior information cited in this file,

it asks:-

```
PEST control file has only one observation.
Minimize model equiv or match to observation? [i/a] (<Enter> if "i"):
```

Enter "i" (or press the <Enter> key) to inform CMAES_P that it must simply lower the value of the single model output corresponding to the single observation cited in the PEST control file. In this case the actual value of that observation, and its weight, listed in the PEST control file is immaterial. On the other hand, enter "a" to instruct CMAES_P to minimize the square of the difference between the single model output and its "observed value" as recorded in the PEST control file.

In all other cases of CMAES_P usage, CMAES_P minimizes exactly the same weighted least squares objective function as PEST does.

Next CMAES_P issues a number of sets of prompts. (Prompts are used rather than an input file as this allows the user to run CMAES_P without actually reading this manual; it also reminds him/her of options available in using CMAES_P. Nevertheless the use of prompts is a little cumbersome and later versions of CMAES_P may provide an input file alternative.) All prompts include a default response; this is accepted simply by pressing the <Enter> key.

The first five prompts issued by CMAES_P are listed below; note that the default values provided below are case-specific.

```
Enter values for following CMA control variables:-

   Population size, lambda (<Enter> if 11):
   Number of parents, mu (<Enter> if 6):
   Recombination weights [Superlin, Linear, Equal] (<Enter> if superlin):
   Random number seed (<Enter> if 1111):
   Read parameter covariance matrix from a file? (<Enter> if "n"):
```

The "population size" and "number of parents" are λ and μ respectively, discussed in the short theoretical overview of the CMA-ES algorithm provided above. The default value for the former is computed as:-

$$\lambda = 4 + 3\ln(n) \qquad (6.1)$$

where n is the number of adjustable parameters. The default μ is computed as:-

$$\mu = \lambda/2 \qquad (6.2)$$

After λ model runs have been carried out, a new set of mean parameter values is computed which, in conjunction with the evolving covariance matrix, is employed for generation of new parameter value realisations. The mean is computed through weighting the parameter sets corresponding to the μ best results. If the "equal" alternative is selected in response to the third of the above prompts, then the weights applied to all of the μ parameter sets employed in computation of this mean are equal. If the "linear" option is selected, weights are computed as (after ordering from lowest to highest objective function):-

$$w_i = \mu + 1 - i \qquad (6.3a)$$

If the "superlin" option is selected, lower objective functions are assigned an even greater weight. Weights are then computed as:-

$$w_i = \ln(\mu+1) - \ln(i) \qquad (6.3b)$$

Any integer can be selected as the random number seed, and supplied to CMAES_P in response to the fourth of the above prompts. Note that the outcome of CMAES_P runs implemented on the same platform will be identical as long as the same random number seed is selected (even if run stopping and restarting is implemented). Selection of different random number seeds results in the internal generation of different parameter set realisations and hence in the following of different optimisation paths.

The parameter covariance matrix option (which pertains to the fifth of the above prompts) will be discussed in the next section.

Next CMAES_P asks (if the number of observations from which the objective function is computed exceeds the number of adjustable parameters);

```
Employ SVD hybridisation?  [y/n] (<Enter> if "n"):
```

The hybridisation scheme is further explained below. At the time of writing, the default is set to "no". However the user is urged to try this scheme at some stage, especially if he/she suspects that derivatives of model outputs with respect to adjustable parameters are not too bad. If a response of "y" is provided to this question, CMAES_P next asks two further questions:-

```
How many trial singular value thresholds? (<Enter> if 3):
Use "soft" or "hard" hybridization?  [s/h] (<Enter> if "s"):
```

These are further explained below.

CMAES_P next asks:

```
Forgive model run failure? [y/n]: (<Enter> if "y"):
```

If a response of "n" is provided to this prompt, then a failure on the part of CMAES_P to read any part of any model output file after completion of any model run will precipitate cessation of CMAES_P execution with an appropriate error message. However if the response to the above prompt is "n", CMAES_P will interpret an error in reading model output files as evidence of an undigestable parameter set. Internally, it will attribute a very high objective function to this parameter set, thus providing a disincentive to the parameter estimation process from generating a similar set of parameters.

CMAES_P's next seven prompts are:-

```
Termination Criteria:-
   Min rel obj fn change over N itns (<Enter> if 1.00000E-03):
   No of itns (N) over which this applies (<Enter> if 40):
   Min rel param change over N itns (<Enter> if 1.00000E-03):
   No of itns (N) over which this applies (<Enter> if 40):
   Rel high-low generated obj fn diff over N itns (<Enter> if 1.0000E-02):
   No of itns (N) over which this applies (<Enter> if 10):
   Maximum number of iterations (<Enter> if 1000):
```

A number of CMAES_P termination criteria are similar to those employed by PEST. However the default values of variables governing those criteria are different from those commonly employed by PEST.

The first two of the above prompts pertain to the objective function. If it fails to fall by a relative amount provided in response to the first of the above prompts over the number of iterations provided in response to the second of the above prompts, then the optimisation process will be deemed to have reached completion.

The third and fourth of the above prompts pertain to parameter changes. If no parameter undergoes a relative change given by the response to the third of the above prompts over a

number of iterations provided in response to the fourth of the above prompts, then the optimisation process will be deemed to have reached completion.

The fifth and sixth of the above prompts pertain to objective functions computed during any iteration on the basis of the λ trial parameter sets generated during that iteration. If the relative difference between the highest and lowest of these objective functions is small over a number of successive iterations, this indicates that the covariance matrix used to generate parameter realisations contains small parameter variances (hopefully as an outcome of convergence to the global objective function minimum). As time progress, it becomes more and more unlikely that even a small lowering of the objective function will occur under these conditions.

The maximum number of iterations is provided in response to the seventh of the above prompts. If this is supplied as zero, then CMAES_P will undertake just one model run, this being on the basis of initial parameter values as supplied in the PEST control file. This can be useful when the PARREP utility is employed to create a new PEST control file on the basis of values optimised from a previous CMAES_P or PEST run. A single model run can then be undertaken on the basis of these optimised parameter values.

As soon as one of its termination criteria is satisfied, CMAES_P will cease execution, documenting the reason why this occurred.

CMAES_P's final prompt is:-

```
Run model with initial parameters? [y/n]  (<Enter> if "y"):
```

While the default response to the above prompt is "y", a user may prefer to answer "n" in circumstances where model run times are long, especially if CMAES_P undertakes model runs in parallel. In the latter case all machines but one stand idol while this first model run is completed. There is no real advantage to undertaking this run except for the fact that it then "sets the standard" with respect to which objective function outcomes of model runs performed under the control of CMAES_P are measured. Improvements of the objective function achieved through the calibration process are thereby readily apparent.

6.5.3 Stopping and Restarting

Like PEST, CMAES_P can be stopped by issuing the PSTOP or PSTOPST command from another command-line window, opened in the same working directory as CMAES_P. Cessation of execution is instantaneous if CMAES_P is run in parallel mode; however orphaned model runs will continue in their own windows unless halted by the user. When undertaking runs in serial, cessation of CMAES_P execution takes place upon completion of the current model run.

When operating in parallel mode, slaves will not automatically shut down if CMAES_P is halted using the PSTOP command (for often a user-instigated stop is followed by a user-instigated re-commencement of the optimisation process). In contrast, if CMAES_P stops of its own accord because a convergence criterion has been met, of if execution is halted using the PSTOPST command, then slaves will automatically shut down.

A stopped CMAES run can be re-started using the "/r" command-line switch. Execution will re-commence at the beginning of the optimisation iteration at which its execution was previously interrupted. The "/j", "/s" and "/d" restart switches available with PEST are not available with CMAES_P.

If the PPAUSE command is issued from another window, CMAES_P execution will pause at

completion of the next model run. This gives the user the opportunity to inspect model output files based on current parameters if he/she so desires. Program execution can be resumed using PUNPAUSE.

6.5.4 Final Model Run

If CMAES_P ceases execution through fulfilment of one of its termination criteria, or if it's execution is halted through typing of the PSTOPST command, it will undertake one final model run on the basis of optimised parameters. Thus all model input files are "primed" with these optimised parameter values, and all model output files contain model-generated quantities computed on the basis of these parameter values.

It is important to note, however, that this functionality is not available when CMAES_P is run in parallel mode and stopped using the PSTOPST command. The reason for this is that while cessation of CMAES_P execution may follow rapidly from invocation of the PSTOPST command, current model runs must proceed to completion unless terminated by the user. Once this is done, a final model run on the basis of optimised parameters can be undertaken using the PARREP utility and then CMAES_P with the maximum number of iterations set to zero, as described above.

6.6 Supplying a Parameter Covariance Matrix

If desired, an initial parameter covariance matrix can be supplied to the CMA-ES algorithm embodied in CMAES_P. This can prove useful for at least the following reasons.

1. A user may desire that initial parameter standard deviations NOT be calculated as one quarter of lower-to-upper parameter bound intervals.
2. When a CMAES_P run is commenced after a PEST run, the parameter covariance matrix computed by PEST can be supplied as the initial parameter covariance matrix for CMAES_P, thus hopefully saving the many model runs required to evolve a suitable covariance matrix.
3. When CMAES_P is run after a previous CMAES_P run, the parameter covariance matrix computed by CMAES_P in the previous process can be supplied as the initial parameter covariance matrix for the next CMAES_P run. Once again, valuable model runs required for covariance matrix evolution can be saved through this strategy.

Upon commencement of execution, CMAES_P's fifth prompt is:-

```
Read parameter covariance matrix from a file? (<Enter> if "n"):
```

If a user responds to this prompt with "y", CMAES_P then asks:-

```
Enter name of parameter uncertainty file:
```

The format of a "parameter uncertainty file" is described in PEST documentation. (At the time of writing, it is actually documented in the addendum to the PEST manual.) This format provides considerable flexibility in provision of a covariance matrix. In particular:-

1. Parameters can be divided into independent groups, and a partial covariance matrix can be supplied for each group. If desired, a covariance multiplier can be provided for each such matrix, this allowing easy adjustment of the magnitude of each matrix.
2. Where some or all parameters are statistically independent of other parameters, individual parameter standard deviations can be supplied.

Where matrices are supplied, PEST matrix file format must be observed; see PEST documentation for further details. Use of this format allows ready manipulation of this matrix by the increasing number of PEST utilities which undertake such manipulation.

It is important to note that where a parameter is log-transformed, the standard deviation and/or covariance matrix elements associated with that parameter must pertain to the log (to base 10) of its values rather than to native parameter values.

A parameter uncertainty file, and matrices cited therein, can be built in a number of ways. As is discussed shortly, CMAES_P writes a pair of such files itself at the end of the optimisation process (or part way through this process if it is prematurely halted by the user). Alternatively, if a CMAES_P run was preceded by a PEST run, the PEST-computed parameter covariance matrix may be supplied as an initial covariance matrix for the use of CMAES_P. However a little cutting and pasting will be required for proper formatting of this matrix. (It is anticipated that a utility will shortly be written to automate this task.)

6.7 SVD-Hybridisation

Suppose that n parameters are being estimated. After n+1 model runs have been completed (and at intervals of approximately n+1 model runs thereafter), CMAES_P has enough information at its disposal to estimate values for an optimum parameter set using a Gauss method. This estimate would be correct if the model were linear, and if its outputs were uncontaminated by numerical noise. In other modelling circumstances such an estimate may not be good at all.

CMAES_P actually accomplishes this gradient-based estimation of optimal parameters using singular value decomposition (SVD). Use of SVD as a computational device actually allows CMAES_P to make a number of such estimates, with these estimates varying according to the number of singular values used in their computation. Where nonlinearity or model numerical noise is high, or where parameter differences among the set of n +1 model runs employed in this calculation do not properly span the full dimensionality of parameter space, use of a high number of singular values will probably yield poor estimates of optimal parameters; in contrast the use of a low number of singular values may not provide enough "parameter resolution" to advance the parameter estimation process, particularly where the optimal parameter set lies within a long narrow valley in parameter space.

In implementing SVD-hybridized parameter estimation, CMAES_P first selects, from among the last n +1 model runs, the parameter set that led to the lowest objective function. It then adopts as "temporary parameters" unit lengths in parameter space along each of the vectors joining this point to the other n points in parameter space that formed the basis of the other n model runs. For each such parameter it then computes an approximation to the derivative of every model output with respect to this parameter as the difference between the respective model output and that for the base model run (this corresponding to the minimum objective function of the n + 1 model runs). Thus it builds a Jacobian matrix **X** for these parameters.

Next CMAES_P undertakes singular value decomposition of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ (where **Q** is the observation weight matrix), defining matrices **V** and **E** in the process such that:-

$$\mathbf{X}^t\mathbf{Q}\mathbf{X} = \mathbf{V}\mathbf{E}\mathbf{V}^t$$

where **V** is the matrix of normalized eigenvectors of **X**, and **E** is a diagonal matrix comprised of singular values of $\mathbf{X}^t\mathbf{Q}\mathbf{X}$ arranged in descending order. A subset of these values comprising a matrix $\mathbf{E}_1$ is then selected, and a parameter upgrade vector computed as:-

$$\mathbf{p} = \mathbf{V}_1\mathbf{E}^{-1}{}_1\mathbf{V}^t{}_1\mathbf{X}^t\mathbf{Q}\mathbf{r}$$

where **r** is the vector of model-to-measurement residuals computed on the basis of the base parameter set, and $\mathbf{V}_1$ is the subset of **V** comprised of eigenvectors corresponding to the subset $\mathbf{E}_1$ or **E**. Actual parameter values are then computed from this temporary parameter set. The parameter upgrade vector is shortened if necessary in order to keep actual parameter values within user-prescribed bounds.

The point at which singular value truncation takes place is determined by singular value ratios. Cutoff ratios of 0.1, 0.01, 0.001 etc (in that order) are selected, depending on the user's choice of number of truncation levels to test. For each such selection a new parameter set is estimated (unless use of two different singular value ratios results in the same number of selected singular values). The objective function corresponding to each new parameter estimate is then computed on the next occasion that a series of parallel runs is undertaken.

CMAES_P provides two options for assimilating the results of SVD-based parameter computation into the overall CMA-ES optimisation process, these being labelled as "soft" and "hard" in its prompts. If the "soft" option is taken, the best of the currently-selected λ parameters (these forming part of the n +1 member parameter set on which SVD analysis was based) is replaced by the SVD-computed parameter set only if the objective function achieved through the SVD process yields the lowest objective function achieved to date. If the "hard" option is selected, parameter set replacement is undertaken if the SVD-computed parameter set leads to a lower objective function than that computed only on the basis of the current λ parameter sets. The former option is likely to be better if the objective function surface is pitted with many local minima, for the latter option may lead to too rapid convergence and hence heighten the chances of being trapped in a local optimum. The latter option may lead to more rapid convergence if the objective function surface is relatively free of local optima.

The "cost" of undertaking SVD-hybridisation is that k extra model runs need to be carried out on every second or third optimisation iteration (depending on the size of λ relative to n) where k is the number of tested SVD truncation levels. If parallelisation of model runs is being undertaken, this may or may not be an issue. Ideally λ should be a multiple of the number of slaves that a user has at his/her disposal in normal CMAES_P operation. When SVD-hybridisation is being employed, $\lambda + k$ should be a multiple of the number of slaves. If there are slaves to spare, then there is no cost whatsoever involved in using SVD-hybridisation, even if no SVD-based attempt at computation of a parameter upgrade yields a better result than that obtained on the basis of the CMA-ES algorithm alone. In other cases, the worth or otherwise of employing SVD-hybridisation will depend on its success in accelerating the optimisation process. In some circumstances the average extra cost per iteration will be more than balanced by the reduced number of iterations required to achieve convergence.

A hidden cost of using the SVD-hybridisation scheme is that CMAES_P memory requirements can be large. This is because parameters, and all model outputs, associated with n +1 model runs must stored in memory. In many cases this is not a high cost; however if many parameters and/or observations are involved in the parameter estimation process, the cost may become high.

At the time of writing, SVD-hybridisation is not allowed if an observation covariance matrix is supplied instead of measurement weights for one or more observation groups.

6.8 CMAES_P Output Files

CMAES_P records the history of its optimisation process both to the screen and to its run record file. The latter has the same filename base as the PEST control file, but possesses an extension of "*.rec*". This file finishes with a table of optimised parameter values.

At any stage of the optimisation process, best parameters achieved to date are available in a "parameter value file". This has the same filename base as the PEST control file, but possesses an extension of "*.par*". Parameter values recorded in this file can be placed into another PEST control file (as initial values recorded in that file) using the PEST PARREP utility. At the end of a CMAES_P run, this procedure can be used to create an input dataset for PEST, in which PEST can then be used to create a series of its own output files based on optimised parameters. To obtain a residuals file set NOPTMAX to zero in the PEST control file so that PEST undertakes just one model run. To obtain parameter covariance and uncertainty data, set NOPTMAX to -1.

CMAES_P records its final covariance data in a parameter uncertainty file. This file has the same filename base as the PEST control file but possesses an extension of "*.unc*". The actual matrix is stored in a file with an extension of "*.cmf*".

On the basis of the parameter value and covariance data files recorded by CMAES_P, it is possible to commence a new CMAES_P run with new control variables (for example λ and μ) while retaining the benefits of progress made during a previous CMAES_P run. A suitable procedure is as follows.

1. Use the PARREP utility to build a new PEST control file from the PEST control file on which the just-completed CMAES_P run was based, but with initial parameter values replaced by values computed by CMAES_P.
2. Commence another CMAES_P run on the basis of this new PEST control file. When CMAES_P prompts for inputs, provide non-default values as desired for variables such as λ and μ. Also inform CMAES_P that a parameter covariance matrix is to be provided; then provide the name of the parameter uncertainty file written by CMAES_P on its previous run.

If appropriate, parameter covariance data can be modified from that recorded in the parameter uncertainty file by CMAES_P on its previous run. Modifications may include the following.

1. The variance multiplier recorded in the CMAES_P-produced parameter uncertainty may be made larger or smaller in order to widen or narrow the initial search region in the re-commenced CMAES_P run.
2. Instead of a covariance matrix, it may be desired that only parameter standard deviations be supplied, these being comprised of the square roots of diagonal elements of the covariance matrix; thus the temporary assumption of parameter independence is made. This can be achieved through use of the PEST MATDIAG utility, together with a little cutting and pasting.

6.9 Parallelisation

As for PEST, CMAES_P-invoked model runs can be undertaken in parallel across computer networks and/or processors. Model runs are carried out through the agency of slaves which receive and send signals to CMAES_P through message files. See the PEST manual for further details.

When parallelisation is implemented, the CMAES_P input dataset must include an additional file, this being the parallel run management file. This should have the same filename base as the PEST control file, but possess an extension of "*.rmf*". This file is identical to the PEST run management file; all variables have the same role, except for the PARLAM variable which, if present, is ignored. It is also important to note that the IFLETYP variable must be set to 0 rather than 1. Thus individual model input and output filenames are not supplied for each slave; rather their names are determined through prefixing of the pertinent slave working directory name to model input and output filenames supplied in the PEST control file.

As for PEST, CMAES_P records a complete history of communications between it and its slaves to a run management record file; this has the same filename base as the PEST control file, but possesses an extension of "*.rmr*" (for "run management record").

The number of runs completed during the current package of parallel run calls (which normally coincides with an iteration, the total size of the package therefore being λ) is available in a file with the same filename base as the PEST control file, but with an extension of "*.rcr*".

There is one major difference between the parallel version of PEST and the parallel version of CMAES_P. Parallel PEST is run as a separate executable (named PPEST). Parallelisation of CMAES_P is activated through a command-line switch, this being the "/p" switch. The same executable is employed for both serial and parallel runs.

The choice to parallelise model runs may have a bearing on the choice of certain CMAES_P control variables. If the parameter population size (λ) is a multiple of the number of slaves, then slaves are not idle for part of an iteration. If SVD-hybridisation is implemented, the number of slaves should be a multiple of the population size plus singular value cutoffs implemented.

6.10 Pseudo-Regularisation

When run in regularisation mode, PEST adjusts weights on an iteration-by-iteration basis for observations and prior information equations belonging to observation groups whose names begin with "regul". Two types of weights adjustment are performed.

1. If the IREGADJ regularisation control variable is set to 1, then weights adjustment between regularisation groups is undertaken, such that those groups which are most sensitive to parameters which are least sensitive to actual measurements receive greater weight.
2. Subsequent to this adjustment, a global weights multiplier is applied, this being equivalent to the Lagrange multiplier employed in the constrained minimisation process through which Tikhonov regularisation is implemented in PEST. In this constrained minimisation problem, the regularisation objective function is minimized subject to the constraint that the measurement objective function is equal to its user-specified target, PHIMLIM.

Such functionality is not available through CMAES_P, for implementation of both of the above weights adjustment strategies requires that sensitivities of model outputs with respect to adjustable parameters be available. However, CMAES_P does not compute these sensitivities. As has already been discussed, one of its strengths is that it can thus perform satisfactorily in a model calibration context in which model outputs are corrupted by

numerical noise. However, as was also discussed above, this comes at a cost. One cost is the fact that model run requirements for objective function minimisation are normally higher than for PEST. Another cost is that implementation of Tikhonov regularisation in the manner discussed above is not possible.

For CMAES_P usage in model calibration contexts this may not be a disadvantage, for normally the number of adjustable parameters will be far fewer than estimated by PEST because of the much higher model run requirements of CMAES_P; hence the need for regularisation will not be as pressing. However in contexts where a management objective function is minimised in order to maximize operational efficiency (i.e. in contexts where CMAES_P is being used for optimisation rather than calibration purposes), the ability to include constraints in the objective function minimisation process may be missed. Certainly the relationships through which constraints are formulated may be included in the CMAES_P objective function. But the ability to compute a weight factor for these constraints equivalent to a Lagrange multiplier is not available. The following procedure may then be worthy of consideration.

1. Set up a PEST control file in which PEST runs in regularisation mode. Set NOPTMAX to 1 in this file, so that PEST runs for just 1 optimisation iteration.
2. Run PEST.
3. Build a new PEST control file. In this file replace weights for all observations and prior information equations belonging to regularisation groups with those recorded in the residuals file on the previous one-iteration PEST run. This file has the same filename base as the previous PEST control file but has an extension of "*.res*" (or "*.rei*" for the intermediate residuals file). These weights have been updated by PEST in accordance with the weights adjustment procedures described above.
4. Instruct PEST that it must run in "parameter estimation" mode.

The new PEST input dataset is then ready to be used by CMAES_P.

Of course the question must be asked "why not use PEST to undertake optimisation – after all, we are attaching credence to its derivatives for the purpose of weights adjustment; why not attach credence to these derivatives for the purpose of constrained minimisation of the regularisation objective function?" This is a good question.

7. A Global Optimiser – SCEUA_P

7.1 Introduction

Like CMAES_P, SCEUA_P is a global optimiser. Like CMAES_P it is interchangeable with PEST in that a PEST input dataset (comprising a PEST control file and template and instruction files cited therein) can also serve as a SCEUA_P input dataset, with no alterations whatsoever required to this dataset. Like CMAES_P, SCEUA_P can work in serial and parallel modes; when used in the latter mode, model runs can be distributed across machines in a network for increased optimisation efficiency. And like CMAES_P, SCEUA_P can be employed to minimise either a least-squares objective function as defined through a PEST input dataset, or an objective function defined in another way by the model.

SCEUA_P implements the SCE ("shuffled complex evolution") algorithm described by Duan (1991) and Duan et al (1992; 1993; 1994); UA stands for "University of Arizona". The SCE method does not require computation of derivatives of model outputs with respect to adjustable parameters. Hence it can operate successfully even where the relationship between model parameters and model outputs is discontinuous, or is contaminated by model-generated numerical noise. It can also perform well where the objective function features local optima on either a large or small scale (or both) in parameter space.

The SCEUA_P program described herein replaces another program of the same name that was previously supplied with the PEST Surface Water Utility suite. The previous program lacked many of the features of this newer version of SCEUA_P, including parallelisation and restart capabilities. It also lacked the ability to accommodate log transformation of parameters, and its screen and run record file reporting of the optimisation process were less detailed than for this newer version. The user should be sure when running SCEUA_P that the new version of this program, and not the old one, is run. If he/she has previously downloaded a PEST Surface Water Utility suite which contains the old SCEUA_P program, then that version of SCEUA_P should be deleted.

The SCE algorithm was developed for calibration of surface water models, where local optima are not an uncommon phenomenon (though it has since been used in many other applications as well). The reader's attention is drawn to the TSPROC program supplied with the PEST Surface Water Utility suite which was also developed specifically for use in the surface water modelling context. This automates preparation of a PEST input dataset (and hence an SCEUA_P input dataset) involving a complex multi-component objective function comprising flows, flow volumes, flow statistics and baseflow-filtered flows.

Code for SCEUA_P encapsulates FORTRAN source code supplied by the University of Arizona. However, as detailed below, the algorithm implemented in that code has been provided with expanded capabilities in its SCEUA_P implementation described herein. It is the author's hope that no bugs were introduced in the enhancement process. If, for a particular application, a user finds that the SCE algorithm as implemented by SCEUA_P does not perform to expectations, then it is not impossible that this is the author's fault.

Another note of caution must be sounded with respect to the parallel version of SCEUA_P described below. If any problems are encountered in using this version (a possible problem is discussed in the pertinent section below), please notify the author so that he can take the necessary steps to rectify the problem in future versions of this program.

7.2 Using SCEUA_P

7.2.1 SCEUA Control Variables

The SCEUA algorithm is described in the abovementioned texts; see, in particular, Duan et al (1992). As for most optimisation algorithms, the method can be "tuned" for optimal performance through adjustment of a number of control variables. Default values are supplied for all of these but 1, this being the number of complexes that are evolved in each evolution step undertaken by the algorithm. Choice of fewer complexes requires the carrying out of fewer model runs; however it raises the chances of failure of the SCE process in finding the global objective function minimum. A value of between 2 and 20 is recommended, with the number increasing with complexity of the problem and the number of parameters being estimated. (Use 5 if in doubt.)

As is discussed in the PEST manual, log-transformation of some or all parameters can make many optimisation problems easier to solve. Parameter transformation is implemented through appropriate setting of the PARTRANS variable in the "parameter data" section of the PEST control file. SCEUA_P also respects the tying and fixing of parameters as implemented in a PEST control file, as well as SCALEs and OFFSETs provided to parameters through this file.

Values for SCEUA_P control variables are provided by the user in response to a series of screen prompts issued by SCEUA_P. This eliminates the need for preparation of a dedicated SCEUA_P input file. On most occasion of SCEUA_P usage, the value of only one control variable need be supplied, this being the number of complexes to employ in the current optimisation process. Through responding to all other prompts simply be pressing the <Enter> key, default values are accepted. In most cases these work well.

7.2.2 Running SCEUA_P

SCEUA_P is run by typing its name at the command-line prompt, followed by the name of the PEST control file which it must read. Optional parallelisation and/or restart switches can follow that. Thus, for example, if the name of the PEST control file for the current case is *case.pst*, then SCEUA_P should be run by typing the following line at the command-line prompt:-

```
SCEUA_P case
```

Note that either the full name of the PEST control file (i.e. "*case.pst*" in the example above), or simply its filename base (i.e. "*case*" in the example above) can be provided on the command line. An optional "/p" switch following the name of the PEST control file signifies parallelisation (see below), while an "/r" switch instructs SCEUA_P to re-start a previous run whose execution was terminated before completion. (Another command line switch, "/s", can also be employed; this is discussed below in conjunction with SCEUA_P parallelisation.)

Immediately upon commencement of execution, SCEUA_P reads the cited PEST control file. If an error exists within this file, the nature of the error is reported to the screen prior to termination of SCEUA_P execution. It is important however that, just as for PEST, the complete input dataset be checked using the PESTCHEK utility prior to running SCEUA_P, for SCEUA_P error checking is not as exhaustive as that of PESTCHEK; certain undetected errors can, in some circumstances, lead to unpredictable SCEUA_P behaviour.

After having read the PEST input dataset, SCEUA_P issues the following series of prompts for values of its control variables. If desired, all but the first can be responded to simply by

pressing the <Enter> key, this signalling acceptance of the displayed default value. Alternatively, the user can provide an appropriate value of his/her choice. However if an inappropriate value is supplied SCEUA_P will respond to the user's input with contempt, repeating the prompt until the user responds with a control variable value that SCEUA_P finds less repugnant.

SCEUA_P's first set of prompts are as follows (note that the default values depicted below are context-specific):-

```
Enter initial number of complexes:
Minimum number of complexes (<Enter> if 5):
Parameter sets in each complex (<Enter> if 9):
Parameter sets per sub-complex (<Enter> if 5):
Include initial parameter set in population?  [y/n] (<Enter> if n):
Evolution steps before shuffling (<Enter> if 9):
Random number seed (<Enter> if 555):
Verbose or non-verbose SCEUA printout?  [v/n]: (<Enter> if "n"):
```

As has been stated above, a suitable response to the first of these prompts is often 5. The SCE algorithm provides the option of decreasing the number of complexes as the optimisation process progresses (this resulting in fewer model runs per complex evolution loop). If such a decrease is desired, provide a number in response to the second of the above prompts that is less than that provided in response to the first of them.

The second, third and fourth of the above prompts are self-explanatory.

As is described by Duan et al (1992), the SCE algorithm commences its operations by generating a number of sets of random parameter values and then undertaking a model run on the basis of each of these. The number of such sets is equal to the number of complexes times the number of parameter sets in each complex. These sets are then grouped into respective complexes after ordering on the basis of objective function value. Parameter value generation in this initial phase of the SCE process takes place on the basis of a uniform probability distribution extending between the upper and lower bound of each parameter, these bounds being supplied in the PEST control file. The user should ensure that these bounds are set no wider than is necessary. Note that if a parameter is log-transformed, the probability distribution on the basis of which its random values are generated is log-uniform rather than simply uniform.

Depending on the user's response to the fifth of the above prompts, the "initial parameter set" (this being comprised of initial parameter values as supplied in the PEST control file) are included in the random parameter collection on which initial complex selection is based.

As is common among global optimisers, random number generation underpins much of the SCE optimisation process. This is both the strength and the weakness of such schemes. On the one hand, judicious use of this parameter selection mechanism minimises the probability of being trapped in a local objective function minimum. On the other hand, because these methods don't "head straight downhill" to the perceived lowest point of the objective function terrain, the optimisation process requires many model runs for its completion (mostly many more model runs than are required by PEST). The optimisation process is repeatable if the same random number generator seed is used in subsequent SCEUA_P runs. However selection of a different seed will result in an alternative optimisation path being taken. Sometimes repetition of the optimisation process using a number of different seeds allows the user to verify that the global objective function minimum has indeed been found.

As is described below, one of SCEUA_P's output files is named *sceout.dat*. This is identical to the file of the same name written by the original SCEUA code as supplied by University of Arizona. This original code provides two different options for the recording of data to this file. For the non-verbose option, only the parameter set pertaining to the best estimate of the global optimum at the end of each shuffling loop is recorded. For the verbose option, the entire sample population is recorded at the end of each shuffling loop. Note, however, that parameters in this file are referred to as "*x1*", "*x2*" etc. Depending on the log transformation status of various parameters, the values recorded in this file can be either native parameter values or the logs (to base 10) of these values; the values of tied and fixed parameters are omitted altogether. (As is discussed below, a somewhat more informative, PEST-compatible, record of the optimisation process is available through the SCEUA_P run record file and its associated parameter value file.)

SCEUA_P's next three prompts pertain to termination of the optimisation process. They are as follows:-

```
Max relative obj fn change over N itns (<Enter> if 1.0000E-02):
No of itns over which this applies (<Enter> if 5):
Maximum number of model runs (<Enter> if 10000):
```

If the default values are accepted, SCEUA_P will cease execution if the objective function is not reduced by more than one percent over five successive iterations (i.e. evolution loops), or if 10000 model runs have been carried out.

If a value of zero is supplied in response to the last of the above prompts, SCEUA_P will carry out just one model run using the initial parameter values supplied in the PEST control file. It will then terminate execution after writing the objective function value corresponding to this parameter set (as well as contributions to this objective function by different observation groups) to the screen and to its run record file. This can be a useful device for ensuring that model input files contain optimised parameter values and that model output files contain predictions generated on the basis of these optimised parameter values. If a previous SCEUA_P run was halted prematurely (but the user nevertheless has confidence that near-optimal parameter values were obtained), or was undertaken in parallel mode, then the PARREP utility can be employed to build a new PEST control file using parameter values computed on this previous run. By then running SCEUA_P with the "maximum number of model runs" set to zero, the desired aim of building model input and output files on the basis of these values is thereby achieved.

7.2.3 SCEUA_P Output

As it runs, SCEUA_P writes the current value of the objective function (and the contributions made to the objective function by different observation groups) to the screen. This information is replicated in the SCEUA_P run record file. The latter has the same filename base as the PEST control file, but possesses an extension of "*.rec*" (like the PEST run record file). The values employed for SCEUA_P control variables for this run are recorded at the top of this file; optimized parameter values are tabulated at the end of the file.

At any stage of the optimisation process, best parameter values achieved to date can be found in a "parameter value file". This has the same filename base as the PEST control file, but possesses an extension of "*.par*". As mentioned above, the PARREP utility can be used to insert these values into a new PEST control file as initial parameter values in that file.

As discussed above, SCEUA_P produces a third file named *sceout.dat*. This contains a

complete record of the SCE optimisation process written in SCE format.

7.2.4 Interrupting a SCEUA_P Run

SCEUA_P execution can be stopped in its tracks by pressing <Ctl-C>. It can be stopped in a less brutal way by typing PSTOP while situated in the same directory as that employed for the current SCEUA_P run, but within another command-line window. If a final model run based on optimised parameters is desired before cessation of execution, use the PSTOPST command instead (unless SCEUA_P is running in parallel mode and therefore cannot write this file for reasons discussed below). Execution of SCEUA_P can be temporarily paused using the PPAUSE command, and recommenced using the PUNPAUSE command. Note that if SCEUA_P is being used in serial mode, none of these commands will take effect until completion of the current model run. If running in parallel mode they will take effect immediately, while the model continues to run in other window(s) and/or on other machine(s).

7.2.5 Restarting a SCEUA_P Run

A prematurely terminated SCEUA_P run can be restarted by adding the "/r" switch to its command line, after the name of the PEST control file which it must read. If execution was halted during the initial phase of SCEUA_P execution prior to the first instance of complex evolution (i.e. at that stage of the process where model runs are simply undertaken on the basis of random parameter values), then recommencement of SCEUA_P execution will actually take place at the start of the entire SCE optimisation process. Otherwise SCEUA_P will re-commence execution at the beginning of the evolution loop in which its execution was previously interrupted.

Note that in spite of the fact that random number generation forms the basis of the SCE algorithm, a stopped and re-started run will follow the same optimisation path as that which would have been followed if no interruption had taken place.

7.2.6 Minimisation of a Non-Least-Squares Objective Function

If there is no prior information in the PEST control file, and if only one observation is featured in this file, SCEUA_P issues an additional prompt to those listed above. It asks:-

```
Minimize model equiv or match to observation? [i/a] (<Enter> if "i"):
```

If "i" (for "mInimize") is selected, SCEUA_P does not compute an objective function through forming the difference between the single model output and its single measured counterpart and multiplying by the square of the weight. Instead it simply minimizes the single model output, which can therefore be an objective function computed by the model in a way that best suits the user's current application. Selection of "a" (for "mAtch") results in normal SCEUA_P operation in which SCEUA_P attempts to minimize the difference between the single model output and its observed counterpart. Note that the lowest objective function achievable through the latter process is zero. The lower limit to a model-computed objective function depends on the way in which this objective function is defined. There is no reason why this cannot be less than zero.

7.2.7 Final Model Run

If the SCE optimisation process runs to completion, or if it is stopped using the PSTOPST command, SCEUA_P will undertake one final model run using optimized parameter values for this run. This does not occur, however, when SCEUA_P runs in parallel mode for reasons

which will be discussed below. In this latter case the PARREP utility can be used to build a new PEST control file whose initial parameter values are in fact optimised parameter values. SCEUA_P can then be used to undertake just one model run on the basis of this parameter set. (Alternatively, the NOPTMAX variable in the PEST control file can be set to zero and PEST can be employed to undertake this single model run.)

7.2.8 Best Parameter Values

As has already been discussed, at any time during a SCEUA_P run the best parameter values achieved so far during that run are available in a parameter value file whose filename base is the same as that of the PEST control file but whose extension is "*.par*". Thus if SCEUA_P execution is terminated through user intervention or model failure, the best parameter values achieved during the run are nevertheless available for later use. The role of the PARREP utility in facilitating the conduction of a single model run on the basis of these parameters has been described. (Sometimes it may be noticed that use of PARREP in this fashion leads to a lower objective function than that listed in the SCEUA_P run record file. Indeed the objective function is reported in the latter file only at the end of each complex evolution loop. If a lower objective function was encountered mid-loop, but prior to interruption of SCEUA_P execution, the parameter set recorded in the parameter value file is immediately upgraded.)

7.3 Parallel SCEUA_P

7.3.1 Principles of Use

Parallelisation of the SCE optimisation process is not as easy as that of PEST, or of the CMA optimisation process. In both of these latter algorithms batches of model runs are intermittently required for which the parameters employed in one such run are independent of those employed in any of the others. Unfortunately this is not the case for the SCE algorithm, for evolution of parameter complexes is an innately serial process in which the parameters that are employed on one run are dependent on the objective function achieved during the previous model run.

Nevertheless, parallelisation is not impossible, for the evolution of complexes is independent and can thus be undertaken for different complexes on different machines. The procedure with which the SCE process begins (in which a suite of model runs is undertaken on the basis of randomly selected parameter values) is also paralleliseable.

Parallelisation at the level of parameter complexes has been implemented in SCEUA_P. However while the use of SCEUA_P in parallel mode is similar to that of PEST (in that a master and multiple slaves are employed for this purpose), there are nevertheless some important differences of which the user must be aware. These are now outlined.

7.3.2 The SCEUA_P Master

Unlike PEST, the program which implements the parallelised version of the SCE algorithm is the same as that which implements the serial version of this algorithm; parallelisation is invoked simply through use of the "/p" switch on the command line. Thus to run SCEUA_P in parallel mode on the basis of a calibration dataset encapsulated in a PEST control file named *case.pst*, SCEUA_P should be run using the command:-

```
sceua_p case /p
```

The restart "/r" switch can be added to the end of this line if desired.

As for PEST and CMAES_P, slave details must be supplied to SCEUA_P through a run management file whose filename base is the same as that of the PEST control file but whose extension is "*.rmf*". The format of this file is the same as that used by PEST. However the PARLAM variable is ignored, and IFLETYP must be set to zero.

7.3.3 Slaves

As for PEST and CMAES_P, slave functionality is implemented in the PSLAVE program. However there are a number of important differences between the tasks required of a slave when serving PEST and CMAES_P on the one hand, and SCEUA_P on the other. In the former cases the slave supervises the carrying out of a single model run at a time. Hence when a user is prompted for the command which the slave must use to run the model, he/she simply supplies the model command. However when PSLAVE serves a SCEUA_P master the command to run SCEUA_P in "slave mode" must be supplied to PSLAVE as the "model command line", for the "model" in this case is actually a special version of SCEUA_P which either undertakes a single model run, or evolves one parameter complex, as required during pertinent stages of the SCE optimisation process. Thus the slave runs SCEUA_P; it is SCEUA_P which actually runs the model (many times).

There are a number of important consequences of this mode of slave operation. Some of these are as follows.

1. When prompted by PSLAVE for the command to run the model, the user must respond with:-

   ```
   sceua_p case /s
   ```

   where *case* is the filename base of the PEST control file – the same file as that supplied on the command-line of the SCEUA_P master. "/s" is a switch which commands SCEUA_P to run in slave mode.

2. Because of this, the PEST control file, and all template and instruction files cited therein, must be copied to all slave directories from the master directory. This differs from the operation of parallel PEST wherein the PEST master writes model input files and reads model output files across the network based on template and instruction files residing only in its own working directory.

Other important consequences of the fact that parallelisation is undertaken on the basis of parameter complexes rather than individual model runs include the following.

1. Once the original set of model runs based on random parameter values has been carried out, there is no advantage to having more than *n* slaves, where *n* is the user-supplied number of complexes that SCE has been instructed use in the optimisation process.
2. On the other hand, if more than *n* slaves are available, there is no advantage in informing SCE that it must use less than *n* complexes.
3. Because the number of model runs required in evolution of different complexes may be different, some slaves may be idle while waiting for the evolution of other complexes to reach completion, prior to the shuffling of parameter sets undertaken by the master program.
4. Because the use of random numbers in an integral component of the SCE optimisation process, the optimisation path taken for a given problem will be different when solved

by SCEUA_P in parallel mode from the path taken by SCEUA_P in serial mode, even if the same random number generator seed is employed in both cases. This is because the random number generator cannot be used to generate parameter values for the same complexes in the same order in both of these two modes. (Nevertheless a restarted parallel run will indeed follow the same optimisation path as an uninterrupted parallel solution process.)

When run in slave mode, SCEUA_P leaves no output files, for only the master program leaves a record of the optimisation process and the objective function and parameter values achieved through this process. The master SCEUA_P program also works very differently when running in parallel mode than when running in serial mode. In fact it writes its own template file (this being called *sce_slave.tpl*) which in turn governs the writing of the SCEUA_P slave input file named *sce_slave.in*. The SCEUA_P slave in turn writes an output file named *sce_slave.out* which is read by the master SCEUA_P program using an instruction file named *sce_slave.ins* written by itself.

7.3.4 Final Model Run in Parallel Mode

Because of the changes made to its normal operations, the parallelised SCEUA_P master program cannot undertake a final model run using optimised parameters upon completion of the optimisation process. This must be done using the PARREP utility as discussed above.

7.3.5 Additional Output Files

In addition to its normal suite of output files, the master SCEUA_P records an additional file named *case.rcr*, where *case* is the filename base of the PEST control file for the current case. This is updated on every occasion that SCEUA_P requests a model run; *case.rcr* simply records the number of model runs undertaken to date. It must be recalled in interpreting this file, that during the parameter complex evolution process, a "model run" is actually an SCEUA_P run undertaken in slave mode. Parameter complex evolution in fact requires many actual model runs, all of these being undertaken by the SCEUA_P slave.

A parallel run management file named *case.rmr* is also written by the master SCEUA_P when running in parallel mode. This provides a complete record of communication between the SCEUA_P master program and each of its slaves.

7.3.6 A Possible Error Message

As stated above, when undertaking parallelised SCE optimisation, it is actually the slave version of SCEUA_P which runs the model. It does this through a system call. However the slave version of SCEUA_P is actually run by PSLAVE through a system call. Thus the model is actually run by a system call nested within a system call.

Normally this situation presents no problems to an operating system. However it has been noticed that if a computer is very busy with other tasks, an error message such as "cannot read model output file as file sharing is not loaded", may be written to the slave window. This occurs because operating system messages regarding the availability (or lack of availability) of files can get overtaken by other messages to read these files.

If this occurs too often, please notify the author (johndoherty@ozemail.com.au); I may then attempt to overcome the problem by building PSLAVE and SCEUA_P into the one executable program so that system calls do not need to be nested. In the meantime, set the value of the RUNREPEAT variable in the PEST run manager file to 1 (see elsewhere in this

addendum for a description of this variable). This will force SCEUA_P to repeat any model run in which this (or any other) error occurs before declaring the problem to be the model's fault and terminating the optimisation process as a consequence.

7.4 Problem Formulation

When using the SCE method, take care to formulate a well posed inverse problem, i.e. a problem for which there is a unique solution. The SCE algorithm has no built-in regularisation; hence it is up to the user to undertake "manual regularisation" by ensuring that an appropriate level of parameter parsimonisation has been undertaken prior to the calibration process, and that not too much correlation exists between parameters that are retained in the process. The performance of SCE (and most parameter estimation methods) deteriorates as the level of parameter correlation increases.

8. References

Bayer, P. and M. Finkel, 2007. Optimization of concentration control by evolution strategies: formulation, application and assessment of remedial solutions.

Christensen, S. and Cooley, R.L. 1999. Evaluation of prediction intervals for expressing uncertainties in groundwater flow model predictions. Water Resources Research, vol35, no 9, pp 2627-2639.

Cooley, Richard L., 2004. A Theory for Modeling Ground-Water Flow in Heterogeneous Media: Reston, Va., U.S. Geological Survey, Professional Paper 1679.

Duan, Q., 1991. A global optimization strategy for efficient and effective calibration of hydrologic models. PhD thesis, Department of Hydrology and Water Resources, University of Arizona, Tuscon, 1991.

Duan, Q., Soorooshian, S. and Gupta, V., 1992. Effective and efficient global optimization for conceptual rainfall-runoff models. Water Resources Research 28 (4), 1015-1031.

Duan, Q., Gupta, V.K. and Sorooshian, S., 1993. A Shuffled Complex Evolution approach for effective and efficient global minimization. Journal of Optimization Theory and its Applications, 76 (3), 501-521.

Duan, Q., Sorooshian, S. and Gupta, V.K., 1994. Optimal use of the SCE-UA global optimization method for calibrating watershed models. Journal of Hydrology, 158 265-284.

Hansen, N. and A. Ostermeier, 2001. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9, 159-195

Hansen, N., S.D. Muller, and P. Koumoutsakos, 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol. Comput*, 9, 159-195.

Moore, C. and Doherty, J., 2005. The role of the calibration process in reducing model predictive error. *Water Resources Research*. Vol 41, No 5. W05050.

Moore, C. and Doherty, J., 2006. The cost of uniqueness in groundwater model calibration. *Advances in Water Resources*. Volume 29, Issue 4, April, pages 605 – 623.

Paige, C.C. and Saunders, M.A., 1982a. LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans. Math Softw, vol8: 43-71.

Paige, C.C. and Saunders, M.A. 1982b. LSQR: an algorithm for spare linear equations and sparse least squares. ACM Trans. Math. Softw. Vol 8: 192-209.

Poeter, E.P. and Hill, M.C., 2007. MMA, A Computer Code for Multi-Model Analysis. U.S. Geological Survey Techniques and Methods 6-E3, 113p.

Vecchia, A.V. and Cooley, R.L., 1987. Simultaneous confidence and prediction intervals for nonlinear regression models with application to a groundwater flow modwl. Water Resources Research, vol23, no. 7, pp1237-1250.