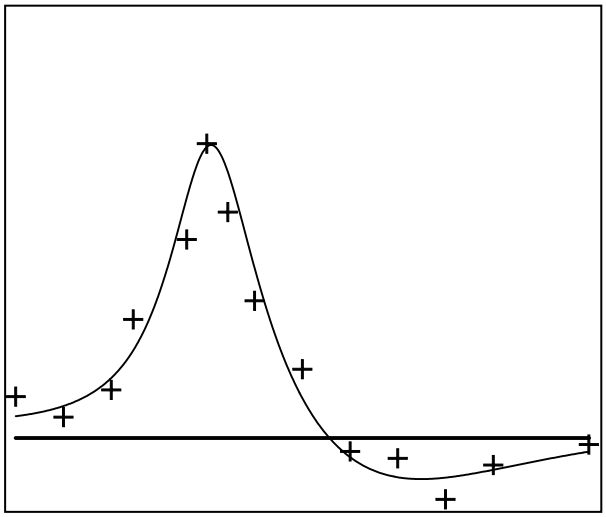
PEST

Model-Independent Parameter Estimation

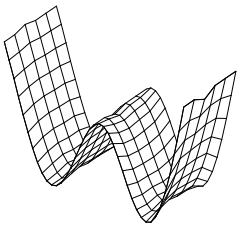
User Manual Part II:

PEST Utility Support Software

(See part I for documentation of PEST, SENSAN and PEST-compatible global optimisers.)



7th edition published in 2018
Latest additions: November, 2026



Watermark Numerical Computing

Disclaimer

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

The author does not make any expressed or implied warranty of any kind with regard to this software. Nor shall the author be liable for incidental or consequential damages with or arising out of the furnishing, use or performance of this software.

PEST Control Variables

See appendix A of part I of this manual for a complete specification of the PEST control file, including a list of all PEST input variables and a description of their roles.

Bugs

If you discover a bug in PEST or any of its utilities, please report it to me, John Doherty, at the following email address.

pestsupport@ozemail.com.au

Table of Contents

1. Introduction 1
2. Files Types and Protocols 2
  2.1 General 2
  2.2 Parameter Value File 2
  2.3 Observation Value File 2
  2.4 Matrix File 3
    2.4.1 General 3
    2.4.2 Specifications 3
  2.5 Uncertainty Files 4
    2.5.1 Introduction 4
    2.5.2 Specifications 5
  2.6 Run Results File 9
  2.7 PAROBS File 10
3. Checking Utilities 11
  3.1 Introduction 11
  3.2 TEMPCHEK 11
  3.3 INSCHEK 12
  3.4 PESTCHEK 13
  3.5 SENSCHEK 14
  3.6 JCOCHEK 15
4. Building and Altering a PEST Control File 16
  4.1 Introduction 16
  4.2 PESTGEN 16
  4.3 TPL2PST 19
    4.3.1 General 19
    4.3.2 The TPL2PST Input File 20
    4.3.3 Blocks and Subblocks 22
    4.3.4 Keywords 22
    4.3.5 Running TPL2PST 24
  4.4 PARREP 24
  4.5 OBSREP 25
  4.6 PARAMFIX 26
    4.6.1 General 26
    4.6.2 The Parameter Fix File 26
    4.6.3 Running PARAMFIX 28
  4.7 ADDPRED1 28
  4.8 ADDREG1 28
  4.9 ADDREG2 29
  4.10 ADDREG2A 31
  4.11 ADDREG3 31
  4.12 ADDCOVMAT 32
  4.13 SUBREG1 33
  4.14 SIMCASE 34

4.15 WTFACTOR....34
4.16 PWTADJ1....35
4.16.1 General....35
4.16.2 Running PWTADJ1....36
4.17 PWTADJ2....37
4.17.1 General....37
4.17.2 Using PWTADJ2....39
4.18 PSTCLEAN....40
5. JCO File Construction and Manipulation....42
5.1 Introduction....42
5.2 JACWRIT....42
5.3 JCO2MAT....43
5.4 MAT2JCO....43
5.5 JROW2MAT....43
5.6 JROW2VEC....44
5.7 JCOL2VEC....44
5.8 JCO2JCO....45
5.9 JCOTRANS....46
5.10 JCOPCAT....47
5.11 JCOMIX....47
5.12 JCOORDER....48
5.13 JCOADDZ....50
5.14 JCODIFF....51
5.15 JCOSUM....51
5.16 JCOCOMB....52
5.16.1 General....52
5.16.2 Using JCOCOMB....53
5.17 JCOSUB....54
5.18 JCOZERO....54
5.19 JCOBLANK....55
5.19.1 General....55
5.19.2 Zero File....56
5.19.3 Using a Jacobian Matrix for Zeroing....57
5.19.4 The JCO Blanking File and JCO Blanking Pattern File....57
5.19.5 The Simultaneous Increment File....58
5.19.6 Running JCOBLANK....58
5.19.7 Observation Weights....59
5.20 JCOCHEK....59
5.21 WTSENOUT....59
5.21.1 General....59
5.21.2 Running WTSENOUT....60
5.22 JCOWT....61
6. Integrity of Finite-Difference Derivatives....62
6.1 Introduction....62
6.2 JACTEST....62
6.2.1 General....62

6.2.2 Using JACTEST 62
6.2.3 Using JACTEST in Parallel Mode 64
6.2.4 Stopping JACTEST 64
6.3 POSTJACTEST 64
6.4 MULJCOSEN 65
7. Model Pre- and Postprocessing 67
7.1 Introduction 67
7.2 PAR2PAR 67
7.2.1 General 67
7.2.2 Using PAR2PAR 69
7.2.3 Using PAR2PAR with PEST 71
7.3 OBS2OBS 75
7.3.1 General 75
7.3.2 OBS2OBS Input File 75
7.3.3 Running OBS2OBS 77
7.3.4 Some Uses of OBS2OBS 78
7.3.5 Some Notes on Using OBS2OBS 79
8. PEST Statistical Postprocessing 81
8.1 Introduction 81
8.2 EIGPROC 81
8.2.1 General 81
8.2.2 Using EIGPROC 81
8.3 PCOV2MAT 83
8.4 INFSTAT 83
8.4.1 General 83
8.4.2 Using INFSTAT 84
8.5 INFSTAT1 86
8.5.1 General 86
8.5.2 Using INFSTAT1 87
8.6 SUPCALC 87
8.6.1 General 87
8.6.2 Using SUPCALC 88
8.7 IDENTPAR 90
8.7.1 General 90
8.7.2 Using IDENTPAR 91
8.8 PARAMID 93
8.8.1 General 93
8.8.2 Using PARAMID 94
8.9 SSSTAT 95
8.9.1 General 95
8.9.2 Theory and Concepts 95
8.9.3 Using SSSTAT 98
8.9.4 Some Further Comments 99
8.10 OBSCOMP 100
8.10.1 General 100
8.10.2 Using OBSCOMP 101

9. Super Parameters and Super Observations 103
9.1 Introduction 103
9.2 SVDAPREP 104
9.3 PCLC2MAT 104
9.4 SUPOBSPREP 105
9.4.1 General 105
9.4.2 Running SUPOBSPREP 106
9.4.3 What SUPOBSPREP Does 107
9.4.4 The New Model 108
9.4.5 Some Features of the New PEST Dataset 109
9.4.6 Using SVD-Assist with Super Observations 109
9.5 SUPOBSPAR 110
9.6 SUPOBSPAR1 111
9.6.1 General 111
9.6.2 Theory 111
9.6.3 Running SUPOBSPAR1 112
10. Linear Error and Uncertainty – Part I 114
10.1 Introduction 114
10.2 SCALEPAR 114
10.2.1 General 114
10.2.2 Running SCALEPAR 115
10.2.3 The New PEST Control File 117
10.2.4 Calculating the Resolution Matrix of Native Parameters 118
10.3 PREDVAR1 119
10.3.1 General 119
10.3.2 Using PREDVAR1 121
10.4 PREDVAR1A 124
10.4.1 General 124
10.4.2 Using PREDVAR1A 124
10.5 PREDVAR1B 125
10.5.1 General 125
10.5.2 Theory 126
10.5.3 Using PREDVAR1B 126
10.6 PREDVAR1C 128
10.6.1 General 128
10.6.2 Theory 128
10.6.3 Using PREDVAR1C 130
10.7 PREDVAR2 131
10.7.1 General 131
10.7.2 Using PREDVAR2 132
10.8 PREDVAR3 134
10.8.1 General 134
10.8.2 Using PREDVAR3 135
10.9 PREDVAR4 137
10.10 PREDVAR5 137
10.10.1 General 137

10.10.2 Using PREDVAR5 139
10.10.3 Observation Weights and Reference Variance 141
10.10.4 Parameter Scaling 141
10.10.5 PREDUNC5 instead of PREDUNC4 141
10.11 PARVAR1 141
10.12 PREDUNC1 142
10.12.1 General 142
10.12.2 Using PREDUNC1 143
10.13 PREDUNC4 144
10.14 PREDUNC5 145
10.15 PREDUNC6 145
10.16 PREDUNC7 146
10.17 GENLINPRED 147
10.17.1 General 147
10.17.2 Tasks Carried out by GENLINPRED 147
10.17.3 Predictive Error Variance and Predictive Uncertainty 148
10.17.4 The Prior Covariance Matrix 149
10.17.5 Observations and Predictions 149
10.17.6 Making GENLINPRED Easier to Use 150
10.17.7 Using GENLINPRED 151
10.17.8 Error and Uncertainty Tables 155
10.17.9 Identifiability 155
10.17.10 Pre- and Post-Calibration Parameter Contributions to Error/Uncertainty 156
10.17.11 PREDUNC Uncertainty Formulation 156
10.17.12 Flexibility 157
10.18 GENLINPRED_ABBREV 157
11. Linear Error and Uncertainty – Part II 158
11.1 Introduction 158
11.1.1 General 158
11.1.2 Concepts 158
11.1.3 Some Special Considerations 161
11.1.4 PEST Requirements 162
11.1.5 Program Versions 163
11.2 RESPROC 163
11.2.1 General 163
11.2.2 What RESPROC Does 163
11.2.3 Running RESPROC 164
11.2.4 SVD-Assisted Inversion 164
11.3 RESWRIT 165
11.4 PARAMERR 165
11.4.1 General 165
11.4.2 Using PARAMERR 165
11.5 PREDERR 166
11.5.1 General 166
11.5.2 Using PREDERR 167
11.6 PREDERR1 168

11.7 PREDERR2 168
11.8 PREDERR3 169
11.9 REGERR 169
11.9.1 General 169
11.9.2 Running REGERR 169
12. Nonlinear Error and Uncertainty 171
12.1 Introduction 171
12.2 RANDPAR 172
12.2.1 General 172
12.2.2 Using RANDPAR 172
12.3 RANDPAR1 174
12.4 RANDPAR2 174
12.5 RANDPAR3 175
12.6 RANDPAR4 176
12.6.1 General 176
12.6.2 Using RANDPAR4 176
12.7 PNULPAR 176
12.7.1 General 176
12.7.2 Using PNULPAR 178
12.7.3 What to do Next 179
12.8 RDMULRES 180
12.8.1 General 180
12.8.2 The RDMULRES Input File 182
12.8.3 Running RDMULRES 183
12.9 MULPARTAB 184
12.9.1 General 184
12.9.2 Using MULPARTAB 184
12.10 COMFILNME 186
12.11 COMFILNME1 187
12.12 REGPRED 187
12.12.1 General 187
12.12.2 Theory 188
12.12.3 Preparing for a REGPRED Run 192
12.12.4 Running REGPRED 193
12.12.5 Some Notes 195
12.12.6 Using the SVD Matrix Option 196
12.13 ASSESSPAR 197
12.13.1 General 197
12.13.2 Theory 198
12.13.3 Using ASSESSPAR 199
13. Latin Hypercube Sampling 201
13.1 Introduction 201
13.1.1 General 201
13.1.2 Using PEST with LHS 201
13.2 PHISTATS 203
13.3 LHS2PEST 204

13.3.1 General 204
13.3.2 LHS2PEST - Level 1 205
13.3.3 LHS2PEST - Level 2 206
13.3.4 LHS2PEST - Level 3 207
13.4 PEST2LHS 208
13.4.1 General 208
13.4.2 Parameter Probability Distributions 208
13.4.3 PEST2LHS Usage Details 209
14. Miscellaneous Utilities 214
14.1 Introduction 214
14.2 PARREDUCE 214
14.2.1 General 214
14.2.2 Theory 215
14.2.3 Using PARREDUCE 218
14.3 CALMAINTAIN 219
14.3.1 General 219
14.3.2 Theory 220
14.3.3 Using CALMAINTAIN 220
14.4 GENLIN 222
14.4.1 General 222
14.4.2 GENLIN Input File 222
14.4.3 Running GENLIN 224
14.5 PESTLIN 225
14.5.1 General 225
14.5.2 Using PESTLIN 226
14.6 DERCOMB1 227
14.6.1 General 227
14.6.2 Using DERCOMB1 228
15. Matrix Manipulation Programs 229
15.1 Introduction 229
15.2 COV2COR 229
15.3 COVCOND 230
15.4 MAT2CSV 230
15.5 MAT2JCO 231
15.6 MAT2SRF 231
15.7 MATADD 231
15.8 MATCOLEX 232
15.9 MATDIAG 232
15.10 MATDIFF 232
15.11 MATINVP 233
15.12 MATJOINC 233
15.13 MATJOIND 233
15.14 MATJOINR 234
15.15 MATORDER 235
15.16 MATPROD 235
15.17 MATQUAD 235

15.18 MATROW 236
15.19 MATSMUL 236
15.20 MATSPEC 236
15.21 MATSVD 237
15.22 MATSYM 237
15.23 MATTRANS 237
15.24 MATXTXI 238
15.25 MATXTXIX 238
15.26 PEST2VEC 238
15.27 VEC2PEST 239
15.28 VECLOG 240
16. RRF and PAROBS Files 241
16.1 Introduction 241
16.2 RRFCAT 241
16.3 RRFCULL 241
16.4 RRFCLEAN 242
16.5 RRF2PAR 242
16.6 PAROBS2RRF 243
16.7 RRF2PAROBS 243
16.8 RRFAPPEND 244
16.9 MULPAROBSTAB 244
16.10 RRF2TAB 244
16.11 RRF2CSV 245
16.12 PARREP_RRF 245
16.13 RRFCALCPSI 245
16.13.1 General 245
16.13.2 Using RRFCALCPSI 246
16.14 RRF2JCO 247
16.14.1 General 247
16.14.2 Theory and Concepts 247
16.14.3 Using RRF2JCO 248
17. Support for the PEST++ Suite 250
17.1 Enhanced Jacobian Matrix Files 250
17.2 EJCOL2VEC 250
17.3 EJROW2VEC 250
17.4 EJROWCOL 251
17.5 GETCSVCOL 251
17.6 ADDCSVCOL 252
17.7 JCB2RRF 252
17.7.1 General 252
17.7.2 Using JCB2RRF 253
17.8 JCB2RRF1 253
17.9 JCB2CSV 254
17.10 CSV2JCB 254
17.11 JCB2PAR 254
17.12 JCB2AVEPAR 255

17.13 JCBCOMB 256
17.14 CSV2PAR 256
17.15 RANDOBS 258
17.15.1 General 258
17.15.2 Running RANDOBS 258
18 Data Space Inversion 260
18.1 DSI1 260
18.1.1 Theory 260
18.1.2 Some Usage Details 262
18.1.3 Using DSI1 265
18.1.4 DSI1 Output File 267
18.1.5 Experience and Recommendations 269
18.2 DSI2, DSIMOD and POSTDSIMOD 270
18.2.1 Background 270
18.2.2 Theory 270
18.2.3 Some Algorithmic Details 271
18.2.4 Running DSI2 272
18.2.5 Supplying Measurement Noise Standard Deviations for PESTPP-IES 275
18.2.6 Negative Weights 277
18.2.7 Running DSIMOD 277
18.2.8 Running PESTPP-IES 277
18.2.9 Calibrating the DSI Model 278
18.2.10 Post-Calibration Uncertainty Analysis using POSTDSIMOD 279
18.2.11 Workflow Summary 283
18.2.12 Other Options 284
18.3 DSI3 285
18.3.1 General 285
18.3.2 The Extra Prompt 285
18.4 DSICSVCLIP 286
18.5 ADDPRED1 287
18.5.1 Background 287
18.5.2 Running ADDPRED1 288
18.5.3 Prior Information Equations 288
18.5.4 The Value of PD0 288
18.6 PDC1 289
18.6.1 General 289
18.6.2 Some Theory 290
18.6.3 Using PDC1 292
18.6.4 The PDC1 Output File 293
18.6.5 Some Notes and Reminders 295
19 DSI by Claude 296
19.1 Introduction 296
19.1.1 The Three Programs 297
19.1.2 The Normal Workflow 297
19.1.3 Command Line and Prompts 297
19.2 Input Files 298

19.2.1 The Training Ensemble (CSV) 298
19.2.2 The Partial PEST Control File 298
19.3 What the Method Computes 298
19.3.1 General 298
19.3.2 The Gain Does Not Depend on the Measurements 299
19.3.3 Conditioning Happens in Whichever Space is Cheaper 299
19.3.4 The Conditioned Ensemble 299
19.4 DSI_SETUP 300
19.4.1 Prompts 300
19.4.2 Outputs 300
19.4.3 The Report File, Section by Section 300
19.5 Normal Score Transformation 303
19.5.1 What it Does 303
19.5.2 What it Fixes and What it Does Not 303
19.5.3 Why There are Two Separate Switches 303
19.5.4 The One Approximation 303
19.5.5 Does it help? Guidance 304
19.6 Subspace Truncation 304
19.6.1 The Problem it Addresses 304
19.6.2 What Truncation Does 305
19.6.3 The Two Controls, and Which to Use 305
19.6.4 Choosing *p* 305
19.6.5 What Truncation Can and Cannot Fix 305
19.6.6 Cost 306
19.7 DSI_PREDICT 306
19.7.1 Inputs and Checks 306
19.7.2 The Results File 307
19.7.3 Where those Numbers Come From 307
19.7.4 The Ensemble File 308
19.8 DSI_XVAL and the Interpretation of Results 308
19.8.1 What it Does 308
19.8.2 The In-Sample Trap 308
19.8.3 The Summary File, Single Dimension 309
19.8.4 The Summary File, Sweep Over Dimensions 309
19.8.5 The Two Coverage Columns 310
19.8.6 The Detail File 310
19.9 A Worked Example, and What it Teaches 311
19.10 Suggested Procedure 312
19.11 Known Limitations 312
19.12 Choosing a PEST-Suite Alternative 313
19.13 A Note on Randomized Maximum Likelihood 313
20. References 315

1. Introduction

This document is the second in a two part manual that describes PEST and its ancillary software. The first part of this manual documents PEST (including Parallel PEST and BEOPEST), the very basic SENSAN sensitivity analyser, and the PEST-compatible CMAES_P and SCEUA_P global optimisers. This second part describes software that complements and supports PEST, and that is distributed with PEST itself.

For the purpose of documentation, the utility programs described herein have been grouped according to function, with separate chapters being devoted to separate functions. Within each chapter programs are grouped according to similarity of task.

In contrast to ensuing chapters, chapter 2 of this manual does not describe any utility programs. Instead it presents protocols for a number of files which are read and/or written by these utilities.

Some of the programs described in this manual have been superseded by others but are still distributed with PEST. These programs to which this applies are identified as such in the description that follows. Their retainment in the PEST suite is based on the fact that newer programs may do things slightly differently, or may omit some of the functionality of the older programs which some users may find attractive. In one or two cases, incompatibilities of which I am unaware may exist between newer versions of PEST and these superseded utility programs. If you discover such incompatibilities, please notify me.

As is the case for part I of this manual, little theory is presented in this document. Instead the reader is referred to the "PEST Book"; this is Doherty (2015), and is downloadable from the PEST web pages. The PEST Book presents and derives equations on which most of the utilities described herein are based. For those rare exceptions where the theory provided in Doherty (2015) does not completely describe the operation of a particular utility program, pertinent equations are presented in this manual.

In addition to the programs described in this second part of version 6 of the PEST manual, two additional suites of utility programs can be downloaded from the PEST web pages at

http://www.pesthomepage.org

These are programs which expedite the use of PEST with groundwater and surface water models. Also downloadable from the above site is the parameter list processor PLPROC which supports parameterisation of complex spatial models. These all have their own documentation that, along with the software itself, is downloadable from the PEST web pages.

2. Files Types and Protocols

2.1 General

This chapter provides specifications for some of the files used by utility programs discussed herein. It is hoped that the collection of these specifications into a single chapter makes reference to these specifications easier, and thus facilitates their construction and manipulation.

2.2 Parameter Value File

The parameter value file is described in section 5.3.2 of part I of this manual. A parameter value file is written by PEST whenever an improved set of parameters is obtained. At any stage of the inversion process it thus contains the best set of parameters computed to date. The expected extension of a parameter value file is "*.par*"; however when PEST is conducting SVD-assisted inversion it also writes a parameter value file containing base parameter values, this having an extension of "*.bpa*".

A number of utility programs documented in this manual read and/or write parameter value files. There may be some occasions where you may need to write a parameter value file yourself for one of these programs to read. Fortunately this is easy, as its specifications are simple. An example of a parameter value file is shown below.

```
single point
   ro1      1.00          1.0          0.0
   ro2      40.0          1.0          0.0
   ro3      1.00          1.0          0.0
    h1      1.00          1.0          0.0
    h2      10.0          1.0          0.0
```

Figure 2.1 A parameter value file.

The first line of a parameter value file cites the PEST character variables PRECIS and DPOINT, the values for which are provided in the "control data" section of a PEST control file. Then follows a line for each parameter. Each line contains a parameter's name, its current value and the values of the SCALE and OFFSET variables for that parameter as supplied in the PEST control file.

Entries on all lines of a parameter value file must be separated by one or more spaces.

If a utility program writes a parameter value file, it normally refers to a PEST control file to obtain values for PRECIS, DPOINT, SCALE and OFFSET. If you write this file yourself, suitable values for these variables are "single", "point", "1.0" and "0.0" as shown in the above figure.

2.3 Observation Value File

An observation value file contains two columns of data. The first is comprised of observation names; in accordance with PEST protocol, these names should be 20 characters or less in length and contain no spaces. The second column contains observation values. One or more spaces should separate entries in each column.

Figure 2.2 illustrates an observation value file.

```
 ar1        1.21038
 ar2        1.51208
 ar3        2.07204
 ar4        2.94056
 ar5        4.15787
 ar6        5.77620
 ar7        7.78940
 ar8        9.99743
 ar9        11.8307
 ar10       12.3194
 ar11       10.6003
 ar12       7.00419
 ar13       3.44391
 ar14       1.58278
 ar15       1.10381
 ar16       1.03085
 ar17       1.01318
 ar18       1.00593
 ar19       1.00272
```

Figure 2.2 An observation value file.

2.4 Matrix File

2.4.1 General

Many of the utility programs documented in this manual read and/or write so-called "PEST matrix files". Some convert Jacobian matrix files (i.e. JCO files) to and from this format so that the Jacobian matrix can be easily manipulated using various matrix manipulation utilities, and/or can be transferred between computing platforms that use different binary protocols. Vectors (which are one-dimensional matrices) also use this format; this applies particularly to predictive sensitivity vectors used in linear uncertainty and error analysis.

2.4.2 Specifications

The specifications of a matrix file are illustrated by example. A PEST-compatible matrix file holding a matrix with three rows and four columns is illustrated in Figure 2.3.

```
   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
```

Figure 2.3 An example of a matrix file.

The first line of a matrix file contains 3 integers. The first two integers (NROW and NCOL) indicate the number of rows and the number of columns in the following matrix. The next integer (named ICODE) is a code, the role of which is discussed shortly.

Following the header line is the matrix itself. The matrix is read row by row, with each row

beginning on a new line. Entries within a line must be separated by one or more spaces; they can wrap to the next line if desired. Because matrix elements are read using FORTRAN list-directed input, a compact option for representation of repeated numbers within a matrix row is available. Thus, for example, the line

```
5*3.0, 3*2.4, 2*4.4
```

is equivalent to the line

```
3.0 3.0 3.0 3.0 3.0 2.4 2.4 2.4 4.4 4.4
```

If ICODE is set to 2, the string "* row names" must follow the matrix. This string must be recorded on the line immediately after the last line of the matrix. On the following and ensuing NROW lines must be recorded NROW character strings; each of these strings is the name associated with the respective row of the matrix. Names must be 20 characters or less in length. The string "* column names" must follow that. NCOL column names must then 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 covariance matrix or a resolution matrix). In this case the string "* row and column names" follows the matrix. 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 figure 2.4. 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
```

Figure 2.4 A matrix file containing a diagonal matrix.

2.5 Uncertainty Files

2.5.1 Introduction

Many of the utility programs discussed herein require that you provide a file containing parameter uncertainties. These can be prior or posterior uncertainties, but are normally the former. Thus this file must contain the contents of the C(**k**) matrix which is referred to extensively in Doherty (2015). This matrix is fundamental to linear parameter and predictive error/uncertainty analyses implemented in utility programs documented in this manual.

Some utilities also require that a C(**ε**) matrix be provided for characterisation of measurement noise; others glean this matrix directly from a PEST control file. As is demonstrated below, for those utilities that require it, an uncertainty file can also be used to specify the stochastic

characteristics of measurement noise.

All of the utility programs described herein which read an uncertainty file also read a PEST control file (and often the Jacobian matrix file associated with that PEST control file as well). This PEST control file defines the inversion or uncertainty analysis problem which respective utility programs must solve. The names of entities cited in an uncertainty file must refer to those cited in this PEST control file. Note, however, that it is not always necessary that all entities cited in an uncertainty file be also cited in the PEST control file. Redundancy of information presented in an uncertainty file is often allowed.

2.5.2 Specifications

Figure 2.5 illustrates an uncertainty file.

```
# 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 COVARIANCE_MATRIX
  file "cov.mat"
  variance_multiplier 1.0
  parameter_list_file "list.dat"
END COVARIANCE_MATRIX

START COVARIANCE_MATRIX
  file "cov1.mat"
  first_parameter kpp1
  last_parameter kpp129
END COVARIANCE_MATRIX

START PEST_CONTROL_FILE
  file "te st.pst"
  variance_multiplier 2.0e2
END PEST_CONTROL_FILE
```

Figure 2.5 Example of an uncertainty file.

An uncertainty file allows the user some flexibility in characterising the uncertainty of a group of entities comprising a vector quantity (for example parameters **k** or measurement errors **ε**). Three such options are presently available. These are

1. a list of individual entity standard deviations;
2. a covariance matrix file; and
3. weights assigned to individual elements listed in the "observation data" section of a PEST control file.

A single one of these options can be used to specify the entirety of a **k** or **ε** covariance matrix. Alternatively, different mechanisms can be used within the same uncertainty file to characterise different parts of the total covariance matrix.

An uncertainty file is subdivided into blocks. Each block implements one of the mechanisms of uncertainty characterisation described above. Collectively all the blocks characterise the uncertainty of a set of entities comprising a vector quantity (for example all parameters that are involved in an inversion or uncertainty analysis process). An uncertainty file can have as many blocks as desired. However the following rules must be observed.

1. Any one uncertainty file can be used to characterize the uncertainty of either parameters or measurements (i.e. 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 a PEST control file which defines a particular inverse problem.
3. The uncertainty of an individual element of an overall $\mathbf{k}$ or $\boldsymbol{\varepsilon}$ vector can be characterised in only one way. Thus, for example, any particular element of a $\mathbf{k}$ or $\boldsymbol{\varepsilon}$ vector cannot be cited in a STANDARD_DEVIATION block of an uncertainty file if it is also cited in a matrix provided through a COVARIANCE_MATRIX block of the same uncertainty file.
4. If a parameter is log-transformed in the current inverse or uncertainty analysis problem (as specified in the PEST control file which governs that problem), then specifications of variance, covariance or standard deviation provided in the uncertainty file must pertain to the log (to base 10) of the parameter. In general, the utilities described in this manual provide no checks for this, for they have no way of knowing the transformation status of parameter variances and standard deviations provided in an uncertainty file; it is thus the user's responsibility to ensure that this protocol is observed.
5. An uncertainty file used for the characterisation of $\mathrm{C}(\mathbf{k})$ cannot include a PEST_CONTROL_FILE block.
6. As presently programmed, an uncertainty file used for characterisation of $\mathrm{C}(\boldsymbol{\varepsilon})$ must not include a COVARIANCE_MATRIX block. Note however that most of the utilities described herein do not actually obtain $\mathrm{C}(\boldsymbol{\varepsilon})$ specifications from an uncertainty file. Instead they obtain it directly from the PEST control file which defines the inverse problem. Hence rarely will you need to include a PEST_CONTROL_FILE block in an uncertainty file; uncertainty files are mainly used for specifying $\mathrm{C}(\mathbf{k})$ rather than $\mathrm{C}(\boldsymbol{\varepsilon})$.

Each block of an uncertainty file must begin with a START line and finish with an END line is illustrated in figure 2.5; 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. 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.

2.5.2.1 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(**k**) or C(**ε**) corresponding to these items are zero. Pertinent diagonal elements of C(**k**) and C(**ε**) are calculated by squaring standard deviations (after multiplication by the *std_multiplier*).

If a parameter is featured in a STANDARD_DEVIATION block but is not featured in the PEST control file which forms the basis of the current inverse problem, it is ignored.

2.5.2.2 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. Programs such as PARAMERR which read this section of an uncertainty file only read the "observation data" section of the PEST control file cited in the PEST_CONTROL_FILE block. For any particular observation cited in this section that is also featured in the current inverse problem (as defined by the PEST control file read by utilities which use an uncertainty file for characterisation of measurement noise), its variance is calculated 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.

As stated above, in contrast to the PARAMERR utility, most utilities described in this manual calculate C(**ε**) directly from the PEST control file which defines the inverse or uncertainty analysis problem which it is their task to solve. Hence for most linear analyses as implemented by utility programs documented herein, an uncertainty file does not need to contain a PEST_CONTROL_FILE block.

2.5.2.3 The COVARIANCE_MATRIX Block

Where a parameter uncertainty file provides one or more covariance matrices, each for a subgroup of **k** which shows within-group parameter correlation, all of these matrices are collectively included in a larger C(**k**) matrix, together with variances calculated from parameter standard deviations supplied in one or more STANDARD_DEVIATION blocks which may also be featured in the parameter uncertainty file. Optionally, all elements of a user-supplied covariance matrix provided through a COVARIANCE_MATRIX block can be multiplied by a factor. This factor (for which the default value is 1.0) is supplied following the *variance_multiplier* keyword.

Different options are available for storage of the matrix housed in the covariance matrix file cited in a COVARIANCE_MATRIX block. Matrix storage may follow the PEST matrix file protocol described in section 2.4. If this is the case, then 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, but each new matrix row must start on a new line. 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. If the matrix file follows this protocol then the COVARIANCE_MATRIX block must be identical in format to the first of the COVARIANCE_MATRIX blocks shown in figure 2.5; it can only feature a *file* keyword and an optional *variance_multiplier* keyword.

The PLPROC parameterization utility supplied with PEST writes matrix files whose format is slightly different from that described in section 2.4. This format retains a three-integer header. The first two numbers in this header must specify the number of rows and number of columns in the matrix. The third number must be "1" or "-1"; 1 indicates that the matrix is non-diagonal while -1 specifies a diagonal matrix. The matrix itself follows this header line. However the matrix is not followed by a list of row and column names; instead the end of the file coincides with the end of the matrix. In this case the COVARIANCE_MATRIX block must adopt either the second or third protocols illustrated in figure 2.5. For the second option the user provides a file in which parameters are listed one to a line. There must be as many such lines are there are rows/columns in the covariance matrix. This information links matrix rows and columns to parameters featured in the current inverse problem. Alternatively, the third protocol can be followed. In this case the COVARIANCE_MATRIX block must contain both of the *first_parameter* and *last_parameter* keywords. These refer back to the PEST control file which defines the current inverse problem. Parameters within this PEST control file between and including the user-nominated *first_parameter* and *last_parameter* parameters are then associated with rows and columns of the covariance matrix, with the ordering of parameters in the matrix file being the same as that in the PEST control file. Naturally, the number of parameters in this implied parameter list must be the same as the number of rows and columns in the covariance matrix.

In all of the above cases the *variance_multiplier* keyword is optional. If omitted, it is assumed to be 1.0.

The following should be noted.

1. A covariance matrix must be positive definite.
2. If the first or second of the above COVARIANCE_MATRIX block protocols is adopted, then the order of rows and columns of the covariance matrix (which corresponds to the order of parameters listed either within the matrix file itself or in a user-supplied parameter list file) is arbitrary. Any of the utility programs documented herein which reads a parameter uncertainty file will re-arrange matrix rows and columns so that they correspond to the order of adjustable parameters supplied in the PEST control file on which the current inverse problem is based.
3. A user-supplied covariance matrix need not describe all of the parameters associated with the current inverse problem, for it need only pertain to a subset of these. Other parameters can be associated with other COVARIANCE_MATRIX blocks and/or can be cited in one or more STANDARD_DEVIATION blocks supplied in the same uncertainty file. However a covariance matrix must not be associated with any parameters which do NOT pertain to the current inverse problem.
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.

2.6 Run Results File

PEST_HP writes a run results file if it is run with the "/f" switch, or if the RRFSAVE control variable is set to "rrfsave". This file records the outcomes of a series of model runs. Parameter values, and the values of model outputs calculated using these parameter values, are recorded for all of these runs.

The run results file is an ASCII (i.e. text) file; figure 2.6 illustrates its contents.

```
* case dimensions
       104        1445
* parameter names
k_ppt1
k_ppt2
..
* observation names
well01_1
well02_1
..
* parameter set index
    1
* parameter values source
file prandom1.par
* parameter values
  0.8218973
   1.136450
..
* model output values
   4.106813
   2.810568
..
* parameter set index
    2
* parameter values source
file prandom2.par
* parameter values
   1.694685
   1.741825
etc
```

Figure 2.6 Part of a run results file.

The first part of a run results file is its "header". This is comprised of three subsections. Each subsection begins with a line on which is recorded the "*" character, followed by a space, followed by the name of the subsection. The first subsection ("case dimension") list the number of parameters and observations that are featured in the run results file. (Note that prior information equations are not featured in a run results file.) Then, in the "parameter names" and "observation names" sections, the names of parameters and observations are listed.

Following the header section, a series of sets of parameter values and corresponding model output values (i.e. the model-generated counterparts to observations) are provided. Each of these sets is endowed with a "parameter set index". Run results file protocol dictates that these indices start at 1, and are incremented by 1 for each parameter and corresponding model output set. Parameter values are listed in the "parameter values" subsection, while model

output values are listed in the "model output values" subsection. A further subsection is entitled "parameter values source". This section must contain a single entry; it allows the program which writes the run results file to record where the parameter values came from. This may assist later processing of the file.

Two model output values have special meaning. If the value of any model output associated with a particular parameter set is -1.1E35, this is indicative of a failed model run. Meanwhile, a model output value of -1.22E35 indicates an abandoned model run. Normally all model outputs associated with a particular parameter set will be endowed with these values if even one of them is.

2.7 PAROBS File

The first part of a PAROBS file is identical to a parameter value file. It lists parameter names and values, together with parameter SCALE and OFFSET values; normally the later are 1.0 and 0.0 for all parameters. As for a parameter value file, the first line of a PAROBS file contains values for the PEST control variable PRECIS and DPOINT.

Immediately following parameter values are model-calculated values for observations. Each line in this section of the file has two entries. The first is an observation name; the second is the model-calculated value of that observation. Model output values of -1.11E35 and -1.22E35 have the same meaning as for a run results file.

Figure 2.7 shows part of a PAROBS file.

```
single point
        k_ppt1     3.95511010          1.000          0.000
        k_ppt2     3.49892220          1.000          0.000
        k_ppt3    0.704972890          1.000          0.000
        k_ppt4    0.658135090          1.000          0.000
        k_ppt5     1.34603540          1.000          0.000
..
      k_ppt103     2.20401640          1.000          0.000
      k_ppt104    5.436436300E-02      1.000          0.000
      well01_1     5.74087200
      well02_1     5.56643400
      well03_1     5.73060000
      well04_1     4.14758000
etc
```

Figure 2.7 A PAROBS file.

3. Checking Utilities

3.1 Introduction

This chapter documents utility programs that can be used to test the integrity of a PEST input dataset, and/or various components of that dataset. They are normally used prior to running PEST, either during the initial stages of inverse problem definition, or later in the process when PEST datasets are manipulated with addition or subtraction of Tikhonov regularisation and/or in preparation for post-calibration linear or nonlinear uncertainty analysis.

A utility for checking a SENSAN input dataset is also described in this chapter.

3.2 TEMPCHEK

Program TEMPCHEK checks that a PEST template file follows the correct protocol. Specifications and syntax for PEST template files are provided in chapter 2 of part I of this manual.

If provided with a set of parameter values, TEMPCHEK can also be used to generate a model input file from a template file. It populates a model input file with parameter values in the same way that PEST does. Once a model input file has been built, the model can be run on the basis of this file. It can then be verified that the model experiences no difficulties in reading a model input file generated by TEMPCHEK, and therefore by PEST.

TEMPCHEK is run using the command

```
tempchek tempfile [modfile [parfile]]
```

where

tempfile	is the name of a template file,
modfile	is the name of a model input file to be generated by TEMPCHEK (optional), and
parfile	is the name of a PEST parameter value file (also optional).

The simplest way to run TEMPCHEK is to use a command such as

```
tempchek model.tpl
```

When invoked in this way, TEMPCHEK simply reads the template file *model.tpl*, checking it for breaches of PEST protocol. It writes any errors that it finds to the screen. If desired, these errors can be redirected to a file using the ">" symbol on the TEMPCHEK command line. Thus to run program TEMPCHEK, directing it to write any errors found in the template file *model.tpl* to the file *errors.chk*, the following command should be used.

```
tempchek model.tpl > errors.chk
```

If no errors are encountered in the template file, TEMPCHEK informs you of this through an appropriate screen message. This message also informs you of the number of parameters that TEMPCHEK identified in the template file. TEMPCHEK lists these parameters in a file named *file*.*pmt*, where *file* is the filename base of the template file which it reads. If the template file has no extension TEMPCHEK simply adds the extension "*.pmt*" to the name of the template file. If desired, this file can be easily modified to become a parameter value file. TEMPCHEK can then be used to generate a model input file on the basis of parameter values which you provide in this modified file (see below).

If a parameter is cited more than once in a template file, the parameter is nevertheless written only once to file *file.pmt*; also, it is counted only once when TEMPCHEK sums the total number of parameters featured in the template file.

If you wish to use TEMPCHEK to generate a model input file you must supply it with the name of the template file upon which the model input file is based, the name of the model input file which it must generate, and the values of all parameters named in the template file. To run TEMPCHEK in this fashion, enter a command such as

```
tempchek model.tpl model.in
```

or

```
tempchek model.tpl model.in pestcase.par
```

The name of the parameter value file is optional. If you don't supply a name, TEMPCHEK generates the name itself by replacing the extension used in the template filename with the extension "*.par*"; if the template file has no extension, "*.par*" is simply appended to the name of the template file. Hence the naming convention of the parameter value file is in accordance with that used by PEST which generates such a file at the end of every iteration of its inversion process. See section 2.2 of this manual for specifications of a parameter value file. TEMPCHEK writes a model input file in an identical fashion to the way that PEST writes a model input file. Thus the PRECIS, DPOINT and parameter-specific SCALE and OFFSET variables cited in a parameter value file have the same roles when this file is used by TEMPCHEK as they do when it is used by PEST.

If TEMPCHEK finds a parameter in a template file which is not listed in the parameter value file which it is asked to read, it terminates execution with an appropriate error message. However the parameter value file can contain more parameters than are cited in the template file; these extra parameters are ignored when generating the model input file. This situation may occur if your model has a number of input files and each has a complimentary template file. A single parameter value file can be used to provide values for parameters cited in all template files.

3.3 INSCHEK

INSCHEK checks PEST instruction files. Like TEMPCHEK it can be used in two modes. In the first mode it simply checks that an instruction file has no syntax errors and respects the specifications set out in section 2.3 of part I of this manual. In its second mode of operation it is able to read a model output file using the directions contained in the instruction file; it then writes a file listing all observations cited in the instruction file together with the values of these observations as read from the model output file. In this way you can verify that not only is your instruction set syntactically correct, but that it reads a model output file in the way it should.

INSCHEK is run using the command

```
inschek insfile [modfile][outfile]
```

where

insfile	is a PEST instruction file,
modfile	is a model output file to be read by INSCHEK (optional), and
outfile	is the name of a file to record observation values.

The simplest way to run INSCHEK is to use a command such as

```
inschek myfile.ins
```

When invoked in this way, INSCHEK simply reads the instruction file *myfile.ins*, checking that every instruction is valid and that the instruction set is consistent. If it finds any errors it writes appropriate error messages to the screen. If you wish, you can redirect this screen output to a file by using the ">" symbol on the command line. Thus to run INSCHEK such that it records any errors found in the instruction file *myfile.ins* to the file *errors.chk*, use the command

```
inschek myfile.ins > errors.chk
```

If no errors are found in the instruction file *myfile.ins*, INSCHEK informs you of how many observations it identified in the instruction set and lists these observations to *file*.*obf*, where *file* is the filename base (i.e. the filename without its extension) of *insfile*; if *insfile* has no extension, the extension "*.obf*" is simply appended to the filename.

For an instruction set to be useable by PEST it must do more than simply obey PEST protocol; it must also read a model output file correctly. You can check this by invoking INSCHEK with a command such as

```
inschek myfile.ins modelout.dat
```

When run in this way, INSCHEK first checks *myfile.ins* for syntax errors; if any are found it writes appropriate error messages to the screen and does not proceed to the next step. Alternatively, if the instruction set contained in *myfile.ins* is error free, INSCHEK reads the model output file *modelout.dat* using the instruction set. If any errors are encountered in this process, INSCHEK generates an appropriate error message and abandons execution; such errors may arise if, for example, INSCHEK finds a blank space where a number should be, encounters the end of the model output file before locating all observations, etc. However if INSCHEK reads the file without trouble, it lists all observations cited in the instruction set, together with their values as read from *modelout.dat* to *file*.*obf*, where *file* is the filename base of *insfile*; in the present example this is file *myfile.obf*. Figure 2.2 shows a typical INSCHEK-generated observation value file.

INSCHEK can be instructed to record observation values that are read from a model output file in a file of the user's choice instead of *file*.*obf*. To activate this option, the name of this file should be supplied as the optional third INSCHEK command line argument.

3.4 PESTCHEK

PESTCHEK should be used when all preparations for a PEST run are complete, i.e. when all template files, instruction files and the PEST control file which "brings it all together" have been prepared. PESTCHEK reads the PEST control file, making sure that all necessary items of information are present in this file and that every item is consistent with every other item (for example that logarithmically-transformed parameters do not have negative lower bounds, that RELPARMAX is greater than unity if at least one parameter is free to change sign during the inversion process, etc.). As PEST does not carry out consistency checks such as these, it is essential that PESTCHEK be used to check all input data prior to running PEST.

PESTCHEK also carries out some of the tasks undertaken by programs TEMPCHEK and INSCHEK in that it checks all template and instruction files cited in the PEST control file for correct syntax. Unlike TEMPCHEK and INSCHEK however, PESTCHEK does not generate a model input file nor read a model output file; nevertheless it does check that all parameters and observations cited in the PEST control file are also cited in the template and instruction files referenced in the PEST control file, and that parameters and observations cited in

template and instruction files are also listed in the PEST control file.

PESTCHEK is run using the command

```
pestchek case
```

where

case is the filename base of a PEST control file.

No filename extension should be provided here; an extension of "*.pst*" is added automatically. This is the same filename base as that which should be provided to PEST on its command line; see section 5.1 of part I of this manual. PESTCHEK reads an identical dataset to PEST.

PESTCHEK writes any errors that it encounters to the screen. If you wish, error messages can be redirected to a file using the ">" symbol on the PEST command line. Thus to check the dataset contained in the PEST control file, *calib.pst*, and the template and instruction files cited therein, directing any error messages to the file *errors.chk*, invoke PESTCHEK using the command

```
pestchek calib > errors.chk
```

If PESTCHEK finds one or a number of errors in your input dataset it is important that you re-run PESTCHEK on the dataset after you have corrected the errors. This is because PESTCHEK may not have read all of your input dataset on its first pass; depending on the errors it finds, it may not be worthwhile (or possible) for PESTCHEK to read an input dataset in its entirety once an error condition has been established. Hence, once you have rectified any problems that PESTCHEK may have identified in your input dataset, you should submit it to PESTCHEK again, being content that the data is fully correct and consistent only when PESTCHEK explicitly informs you that this is the case.

Optionally, PESTCHEK can be run using a command line switch. If PESTCHEK is run using the command

```
pestchek case /s
```

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 messages, for these too are suppressed when it is run with the "/s" switch.

3.5 SENSCHEK

SENSAN, described in chapter 17 of part I of this manual, is a basic PEST-compatible sensitivity analyser. It has its own control file which cites template and instruction files which are used to respectively write model input files and read model output files. Parameter vales used as a basis for sensitivity analysis are provided to SENSAN through a parameter variation file.

SENSAN checks the contents of a SENSAN control file, and all other files cited therein, for correctness and consistency. It is run using the command

```
senschek sencase
```

where *sencase* is the name of a SENSAN control file. If the latter possesses an extension of "*.sns*", then this extension can be omitted from the filename.

SENSCHEK writes its error messages to the screen. It is important to note that if it detects certain errors early in the SENSAN control file, it may not proceed with its checking of the remainder of this file, nor of the template and instruction files cited in the SENSAN control file, nor of the parameter variation file pertaining to the current case. Thus it is important to ensure that once a SENCHEK-identified error has been rectified, SENSCHEK is run again. Only when SENSCHEK explicitly informs you that no errors have been detected in the entire SENSAN input dataset is it safe to run SENSAN.

3.6 JCOCHEK

Program JCOCHEK reads a PEST control file and a corresponding Jacobian matrix file (i.e. JCO file), checking that the two are compatible with each other. It is run using the command

```
jcochek case
```

where *case* is the filename base of the PEST control file. JCOCHEK reads *case.pst* as well as *case.jco*, checking that all adjustable parameters 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 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 about the absence of prior information.
2. If prior information is cited in both files, JCOCHEK does not check that sensitivities recorded in the JCO file are the same as respective prior information parameter coefficients recorded in the PEST control file; it issues a warning to this effect.

4. Building and Altering a PEST Control File

4.1 Introduction

Some of the utility programs described in this chapter are used repeatedly during the parameter estimation and uncertainty analysis processes. PARREP, PWTADJ1, PWTADJ2, ADDREG1, SUBREG1 and WTFACTOR are examples of these. PESTGEN is an older program whose role has largely been supplanted by members of the PEST Groundwater and Surface Water Utility suites. The latter can build complex PEST input datasets which include spatial parameters such as pilot points, and time-series observations such as river flow and quantities derived therefrom. The OBSREP utility can be used in conjunction with PARREP to build a PEST input dataset comprised of calibrated parameters together with model-calculated observations based on these parameters. This can provide a starting point for model-based hypothesis testing. PARAMFIX and SIMCASE are older programs that may still find some use in PEST input dataset manipulation.

The TPL2PST utility builds a "partial PEST control file" in which only a limited number of sections of an overall PEST control file are populated – in particular, those pertaining to parameters. Once a user has constructed template files for a PEST run, TPL2PST facilitates manual preparation of the rest of a PEST input dataset, especially when used in combination with PESTsupport programs such as OLPROC.

4.2 PESTGEN

PESTGEN generates a PEST control file. In most cases this file will need to be modified before PEST is run, as PESTGEN generates default values for many of the PEST input variables recorded in this file; it is probable that not all of these default values will be appropriate for your particular problem.

PESTGEN is run using the command

```
pestgen case parfile obsfile
```

where

case	is the case name. No filename extension should be supplied; PESTGEN automatically adds the extension "*.pst*" to *case* in order to form the filename of the PEST control file which it writes.
parfile	is a parameter value file, and
obsfile	is an observation value file.

Specifications of a parameter value file are provided in section 2.2 of this manual while specifications of an observation value file are provided in section 2.3. The former file must include all parameters used in the current case; these parameters may be cited in one or a number of template files. Similarly, the observation value file must provide the names and values of all observations used in the current problem; the observations may be cited in one or a number of instruction files. The observation values provided in this file may be field/laboratory measurements or, if PEST is being run on theoretical data, model-generated observation values. In the latter case INSCHEK may be used to generate the file; if there are multiple model output files, observation value files generated on successive INSCHEK runs can be concatenated to form an appropriate observation value file to provide to PESTGEN.

PESTGEN commences execution by reading the information contained in files *parfile* and

obsfile, checking this information for correctness and consistency. If there are any errors in either of these files, PESTGEN lists these errors to the screen and terminates execution. Alternatively, if these files are error-free, PESTGEN then generates a PEST control file.

Files *parfile* and *obsfile* provide PESTGEN with the names of all parameters and observations which need to be listed in the PEST control file. They also provide PEST with initial parameter values (these must be provided in the second column of the parameter value file), the SCALE and OFFSET for each parameter (in the third and fourth columns of the parameter value file), the laboratory or field measurement set (in the second column of the observation value file), and values for the variables PRECIS and DPOINT (on the first line of the parameter value file). For all other variables listed in the PEST control file, PESTGEN uses default values.

For the parameter and observation value files shown in figures 2.1 and 2.2, the PESTGEN-generated PEST control file is shown in figure 4.1.

```
pcf
* control data
restart estimation
    5    19     5     0    1
    1     1 single point 1 0 0
  5.0   2.0   0.3  0.03    10
  3.0   3.0 0.001
  0.1
   30  0.01     3     3  0.01     3
    1     1     1
* parameter groups
ro1  relative 0.01  0.0  switch  2.0 parabolic
ro2  relative 0.01  0.0  switch  2.0 parabolic
ro3  relative 0.01  0.0  switch  2.0 parabolic
h1   relative 0.01  0.0  switch  2.0 parabolic
h2   relative 0.01  0.0  switch  2.0 parabolic
* parameter data
ro1  none relative    1.00000   1.00000E+10   1.00000E+10 ro1   1.0000   0.00000 1
ro2  none relative    40.0000   1.00000E+10   1.00000E+10 ro2   1.0000   0.00000 1
ro3  none relative    1.00000   1.00000E+10   1.00000E+10 ro3   1.0000   0.00000 1
h1   none relative    1.00000   1.00000E+10   1.00000E+10 h1    1.0000   0.00000 1
h2   none relative    10.0000   1.00000E+10   1.00000E+10 h2    1.0000   0.00000 1
* observation groups
obsgroup
* observation data
ar1    1.21038      1.0  obsgroup
ar2    1.51208      1.0  obsgroup
ar3    2.07204      1.0  obsgroup
ar4    2.94056      1.0  obsgroup
ar5    4.15787      1.0  obsgroup
ar6    5.77620      1.0  obsgroup
ar7    7.78940      1.0  obsgroup
ar8    9.99743      1.0  obsgroup
ar9    11.8307      1.0  obsgroup
ar10   12.3194      1.0  obsgroup
ar11   10.6003      1.0  obsgroup
ar12   7.00419      1.0  obsgroup
ar13   3.44391      1.0  obsgroup
ar14   1.58278      1.0  obsgroup
ar15   1.10381      1.0  obsgroup
ar16   1.03085      1.0  obsgroup
ar17   1.01318      1.0  obsgroup
ar18   1.00593      1.0  obsgroup
ar19   1.00272      1.0  obsgroup
* model command line
model
* model input/output
model.tpl  model.inp
model.ins  model.out
* prior information
```

Figure 4.1 A PEST control file generated by PESTGEN.

Figure 4.1 shows the default values used by PESTGEN in generating a PEST control file. The following features of this file, in particular, should be noted.

- PESTGEN assumes that PEST will be run in "estimation" mode. Neither a "predictive analysis" nor a "regularisation" section is included in the PEST control file.
- PESTGEN generates a separate parameter group for each parameter; the name of the group is the same as that of the parameter. For each of these groups derivatives are calculated using a relative increment of 0.01, with no absolute lower limit provided for this increment. At the beginning of the inversion process, derivatives will be calculated using forward parameter differences, switching to the three-point

"parabolic" method on the iteration following that for which the objective function fails to undergo a relative reduction of at least 0.1 (this being the value that PESTGEN gives to the PHIREDSWH control variable). The derivative increment to be used in implementing the "parabolic" method is twice the increment used in implementing the forward difference method of derivatives calculation.

- No prior information is supplied.
- No parameters are tied or fixed; no parameters are log-transformed and changes to all parameters are relative-limited (with a RELPARMAX value of 3.0). The upper bound for each parameter is provided as 1.0E10, while the lower bound is -1.0E10. It is strongly suggested that you modify these bounds to suit each parameter. It is also recommended that you consider log-transforming some (or all) parameters for greater inversion efficiency. Note, however, that the lower bound of a log-transformed parameter must be positive and that its changes must be factor-limited.
- All observations are provided with a weight of 1.0.
- PESTGEN assumes that the model is run using the command "model". It also assumes that the model requires one input file, namely *model.inp*, for which a template file *model.tpl* is provided. It further assumes that all model-generated observations can be read from one output file, namely *model.out*, using the instructions provided in the instruction file *model.ins*. You will almost certainly need to alter these names. If there are, in fact, multiple model input and/or output files, don't forget to alter the variables NTPLFLE and NINSFLE in the "control data" section of the PEST control file.
- The PESTGEN default values for all other PEST control variables can be read from figure 4.1.

Once you have made all necessary changes to the PESTGEN-generated PEST control file, you should check that your input dataset is complete and consistent using program PESTCHEK. If PESTCHEK informs you that all is correct, then you can run PEST.

4.3 TPL2PST

4.3.1 General

TPL2PST builds a "partial PEST control file" from the contents of one or a number of template files.

When assembling a PEST input dataset for calibration of a complex model, template files are often built first. If these pertain to a groundwater model, some of these template files may be based on pilot point parameters. As such, they may correspond to PLPROC input files. Collectively, these files may feature hundreds (or even thousands) of parameters. TPL2PST can read all of these template files. It then builds a PEST control file in which the following sections are populated. (Headers are provided for other sections of this PEST control file, even though they are empty.)

- control data
- singular value decomposition
- parameter groups
- parameter data
- model input/output

The partial PEST control that is written by TPL2PST will require manual editing. Furthermore, its other sections must be populated in other ways. (This can be done with the help of PEST support programs such as OLPROC, PESTPREP1 and PESTPREP2; the latter two programs belong to the PEST Groundwater Utility Suite.) However, because it populates the "control data" section of the partial PEST control file, and because the "parameter groups" and "parameter data" sections of this same partial PEST control file include all parameters (and groups to which they are assigned) that are cited in one or a number of template files, its use provides an easy and less error-prone alternative to manual construction of these sections of a PEST control file.

4.3.2 The TPL2PST Input File

TPL2PST's tasks are specified by the contents of its input file. This file is easily prepared using a text editor. Figure 4.2 provides an example. Note that blank lines are ignored. Also, any line that begins with the "#" character is treated as a comment line, and hence is also ignored.

```
# An example TPL2PST input file

start template_file
  template_file     = pp1.tpl
  model_input_file  = pp1.dat
  transform         = log
  value             = 1.0
  lower_bound       = 0.1
  upper_bound       = 10
  pargroup          = hk
end template_file


start template_file
  template_file      pp_zone2.tpl
  model_input_file   pp_zone2.dat
  transform          log
  value              5.0
  lower_bound        1.0
  upper_bound        10.0
  pargroup           kgen

  start parameter_type
    text              = kh
    transform         = log
    value             = 5E-03
    lower_bound       = 1E-03
    upper_bound       = 1E-2
    pargroup          = khor2
  end parameter_type

  start parameter_type
    text              = kz
    transform         = none
    value             = 5E-7
    lower_bound       = 1E-7
    upper_bound       = 1E-6
    pargroup          = kvert2
  end parameter_type

end template_file


start template_file
  template_file      pp_zone6.tpl
  model_input_file   pp_zone6.dat

  start parameter_type
    text              = kh
    transform         = log
    value             = 5E-03
    lower_bound       = 1E-03
    upper_bound       = 1E-2
    pargroup          = khor6
  end parameter_type

  start parameter_type
    text              = kz
    transform         = none
    value             = 5E-7
    lower_bound       = 1E-7
    upper_bound       = 1E-6
    pargroup          = kvert6
  end parameter_type

end template_file
```

Figure 4.2 Example of a TPL2PST input file.

As is apparent from figure 4.2, the TPL2PST input file is subdivided into blocks. Each block must begin with the text “start template_file”. Each block must terminate with the text “end template_file”. As is implied by the name of the block, each block pertains to a single template file.

4.3.3 Blocks and Subblocks

Within each block there may be subblocks. This allows parameters within the same template file to be subdivided into types. Parameters will then be ordered by type in the PEST control file that TPL2PST writes. That is, all parameters of one type will be recorded in the PEST control file before those of another type. This can be useful where, for example, a template file corresponds to a PLPROC input file. Perhaps pilot point parameters pertaining to horizontal hydraulic conductivity reside in the 4th column of this file, while those pertaining to vertical hydraulic conductivity reside in the 5th column of this file. In the PEST control file that is written by TPL2PST, all of the horizontal hydraulic conductivity parameters will be cited first; then the vertical hydraulic conductivity parameters will be cited next. This makes it easier to edit the TPL2PST-generated PEST control file.

A "parameter_type" subblock commences with the text "start parameter_type" and finishes with the text "end parameter_type". Multiple "parameter_type" subblocks can exist within the same "template_file" block. Hence multiple parameter types may occur within the same template file. As is discussed below, these are recognized by the occurrence of a specified text string within parameter names.

To preserve compatibility with previous versions of TPL2PST, subblock definition can occur in a number of ways. These are as follows.

1. A "template_file" block may contain no subblocks. In this case, all parameters within the pertinent template file are presumed to belong to a default parameter type which is specific to that "template_file" block. There is no need to provide a name recognition text string for this default parameter type.
2. A "template_file" block may contain multiple "parameter_type" subblocks, but may not include any information pertaining to a default parameter type. It is therefore presumed that all parameters within the template file belong to one of the parameter types which "parameter_type" subblocks define.
3. A "template_file" block may contain multiple subblocks, together with data for a default parameter type. Any parameter that is found in the pertinent template file that is not assigned to one of these subblock-specific parameter types is therefore presumed to belong to the default parameter type that is associated with the overall "template_file" block.

A parameter type is identified by the occurrence of a text string within its name. This is provided with the "text" keyword that is associated with every "parameter_type" subblock. Suppose, for example, that this text string is "kx". Then the occurrence of a "kx" character string in the name of any parameter that is found in the template file that is associated with the "template_file" block to which the "parameter_type" subblock belongs relegates that parameter to the pertinent parameter type. (Note that the same text string can be used to separate parameters into different types in different "template_file" blocks.)

4.3.4 Keywords

Within each block or subblock, data entry follows the keyword protocol. First a keyword is provided. The value of a variable that is associated with that keyword follows. Optionally, a "=" symbol may separate the two.

Each "template_file" block must contain a single "template_file" keyword and a single "model_input_file" keyword. These keywords must NOT appear within a "parameter_type" subblock. The first of these keywords informs TPL2PST of the name of the template file that

is associated with the "template_file" block. TPL2PST must read this file in order to obtain the names of all parameters that then become associated with that block. The second keyword informs TPL2PST of the model input file that corresponds to the template file. Both of these filenames are recorded in the "model input/output" section of the partial PEST control file that TPL2PST writes.

As is apparent from the example in figure 4.2, each "parameter_type" block must contain the following keywords. These can also appear within the general "template_file" block to which the "parameter_type" block belongs; in the latter case they pertain to the default parameter type for that "template_file" block (if a default parameter type exists). Any "parameter_type" subblock must contain all of these keywords. If any of them are present outside of a "parameter type" block in the main part of the "template_file" block, then all of them must be present. These keywords are:

- transform
- value
- lower_bound
- upper_bound
- pargroup

The values that are assigned to these keywords are transferred to the "parameter data" section of the partial PEST control file that TPL2PST writes. All parameters of the same type are assigned the same initial values, lower and upper bounds, transformation type and parameter group. You can alter these in the PEST control file that TPL2PST writes if you wish.

The value supplied for "transform" must be "none", "log" or "fixed". "Tied" is not allowed; if you wish to tie one parameter to another, you must do this yourself in the PEST control file that TPL2PST writes.

The name of a parameter group must be supplied through the "pargroup" keyword. This name cannot be repeated in any other block or subblock. Thus the partial PEST control file that TPL2PST writes will feature as many parameter groups as there are "parameter_type" subblocks (including default parameter type subblocks) in all "template_file" blocks within its input file. (This makes it easy for the ADDREG1 and ADDREG2 utilities to ascribe prior information equations to different parameter groups that are based on different parameter types. This, in turn, allows easy assignment of covariance matrices to different prior information groups. As is stated in documentation for ADDREG1 and ADDREG2, it is wise to limit parameter group names to 6 characters or less in length so that when they are prefixed with the string "regul", the 12 character limit on observation group name length is respected. This constraint does not apply, however, when using ADDREG2A to add regularisation to a PEST control file.)

Note the following:

- Keywords can be provided in any order within a block or subblock.
- If the value associated with a keyword is a text string, then it can be enclosed in quotes. This is mandatory if it is the name of a file and that name contains a space.
- No two "template_file" blocks can cite the same template file. Nor can they cite the same model input file.
- TPL2PST checks for data inconsistencies; it reports them as error messages. Thus, for example, if you supply "transform" as "log" and provide "value" as a negative number, TPL2PST will terminate execution with an explicatory error message.

- If the lower bound provided in a "parameter_type" block is positive, TPL2PST sets the PARCHLIM variable for all parameters of that type in the block-associated template file to "factor". Otherwise it is set to "relative". Alter these in the TPL2PST-generated partial PEST control file if they are inappropriate.
- TPL2PST objects if the same parameter is cited in more than one template file. However it will raise no objections if the same parameter is cited many times in a single template file.
- In writing the "parameter groups" section of its partial PEST control file, TPL2PST provides values for variables which affect finite-difference derivatives calculation. Alter these if they are inappropriate.
- When writing the "control data" section of its partial PEST control file, TPL2PST provides values for the NPAR, NPARGP and NTPLFLE variables which are consistent with information that it has obtained from its input file, and from template files cited therein. NOBS and NOBSGP are set to 0 as the "observation group" and "observation data" sections of the partial PEST control file that it writes are empty.

4.3.5 Running TPL2PST

TPL2PST is run using the command

```
tpl2pst infile pestfile
```

where

infile	is the name of its user-prepared input file, and
pestfile	is the name of the partial PEST control file which it writes.

TPL2PST does not insist that the partial PEST control file have an extension of "*.pst*". However it is a good idea to provide this extension in order to distinguish it from other file types.

4.4 PARREP

Program PARREP replaces initial parameter values provided in a PEST control file by another set of values, the latter being supplied in a PEST parameter value file. See section 2.2 of this manual for specifications of a parameter value file.

Recall from Section 5.3.2 of part I of this manual that in the course of the inversion process PEST writes a parameter value file every time it improves its parameter estimates. After a PEST run has finished (either of its own accord or manually halted), optimised parameter values can be found in the parameter value file. The parameter value file possesses the same filename base as the PEST control file but has an extension of "*.par*". Because it has such a simple structure, a parameter value file can also be easily built by the user with the help of a text editor.

PARREP is useful when commencing a new PEST run where an old run finished. An updated PEST control file can be produced by replacing parameter values in the old file with the best parameter values determined during the previous PEST run as recorded in the parameter value file written during that run. Recommencing a PEST run in this way, rather than through use of the "/r", "/j", "/s" or "/d" switches, allows you to alter certain PEST control variables, fix or tie certain parameters, or adjust PEST's management of the parameter estimation process in other ways, prior to commencement of the new run.

PARREP is also useful when undertaking a single model run on the basis of a certain set of

parameters in order to calculate the objective function. Simply create a new PEST control file using PARREP as described above, and set NOPTMAX to zero in that file.

PARREP is run using the command

```
parrep parfile pestfile1 pestfile2 [new_noptmax]
```

where

parfile	is the name of a parameter value file,
pestfile1	is the name of an existing PEST control file,
pestfile2	is the name for the new PEST control file, and
new_noptmax	optionally provides a new value for NOPTMAX.

When PARREP replaces parameter values in the existing PEST control file by those read from the parameter value file, it does not check that each parameter value lies between its upper and lower bounds, that log-transformed parameters are positive, etc. Hence, especially if using a manually-created parameter value file, it is a good idea to run PESTCHEK before running PEST to ensure that all is consistent and correct.

A special aspect of PARREP's behaviour is worthy of note. If a parameter is tied or fixed in the existing PEST control file which PARREP reads, PARREP will not object if that parameter is omitted from the parameter value file that is provided to PARREP. 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 omitted from the parameter value file 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.

4.5 OBSREP

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 run. 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 calculated on the basis of the new PEST control file should be zero.)

OBSREP is run using the command

```
obsrep resfile pestfile1 pestfile2
```

where

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

4.6 PARAMFIX

4.6.1 General

PARAMFIX can be used to modify complex PEST control files (such as may be constructed for the use of PEST in "regularisation" mode), saving you the time (and propensity for error) that would result from making such file modifications by hand. Alterations facilitated by the use of PARAMFIX are those associated with the introduction of "outside information" on parameter values to the inversion process. Two methods of using such information can be accommodated, the first being the fixing of certain parameters, and the second being introduction of preferred values for certain parameters through prior information. Where many parameters are being estimated, and where prior information equations pertaining to many or all of these parameters are already present within a PEST control file, alteration of an existing PEST control file to accommodate the use of this "outside information" can be a very tedious process if performed manually. PARAMFIX removes much of this tedium.

Use of PARAMFIX is particularly convenient where PEST is being used to calibrate a spatial model (for example a groundwater model) and parameterisation of the model domain is undertaken through the use of pilot points (see manuals of the PEST Groundwater Data Utilities and PLPROC for more details). Thanks to the use of PEST's regularisation functionality, many parameters can be estimated through this process (thus allowing the model to accommodate the spatial heterogeneity that is a fundamental part of most natural systems), numerical stability being maintained through the use of a set of "regularisation constraints". The latter can be supplied as either observations or as prior information, and can take many forms. One form is as a series of "uniformity conditions" in which the parameter value assigned to each pilot point is linked to those of many or all of its neighbouring points through a set of prior information equations expressing the desire that pertinent parameter value differences are zero (heterogeneity is thus introduced to the model domain only where necessary to achieve model calibration). In some modelling contexts, prior information equations expressing this condition throughout the model domain can number in the thousands, with each such equation involving just two parameters. If the user then decides that a particular parameter should be fixed, all prior information equations citing that parameter must be either deleted or modified, for prior information cannot be supplied for parameters which are not adjusted through the inversion process. If the required modifications to prior information equations are done by hand, the chances of making a serious error are enormous.

While PARAMFIX carries out limited checking of the PEST control file which it must modify, its error checking is not complete; it simply assumes that the input PEST control file is correct. Hence PESTCHEK should be used to validate the input PEST control file before running PARAMFIX. If this is not done, and if there is a problem with the PEST control file read by PARAMFIX, the outcome of the control file modification process undertaken by PARAMFIX will be unpredictable. Similarly, after PARAMFIX has written its new PEST control file, the latter should also be checked with PESTCHEK for, under certain circumstances it is not impossible for PARAMFIX to introduce small inconsistencies into this file.

4.6.2 The Parameter Fix File

PARAMFIX requires two input files. One of these is an existing PEST control file (which it modifies and re-writes to a file of a different name); the other is a "parameter fix file" which

contains the information required by PARAMFIX upon which to base its modifications of the existing PEST control file. The parameter fix file has a simple format, and can be prepared using a text editor. A parameter fix file is illustrated in Figure 4.3.

```
ro5     fix_param     10.0   retain_prior
ro2     fix_param     100.0  remove_prior
ro7     fix_param     5.0    remove_prior
ro6     fix_param     6.0    retain_prior
thick1  prior_info    4.0    log          5.0   group1
thick2  prior_info    4.0    none         5.0   group2
```

Figure 4.3 A parameter fix file.

Each line of a parameter fix file contains either 4 or 6 entries. The first entry must be the name of a parameter involved in the current parameter estimation process. This parameter must feature in the "parameter data" section of the PEST control file on which that process is based. If it does not, PARAMFIX will cease execution with an appropriate error message.

As mentioned above, there are two ways in which "outside information" pertaining to a parameter can be introduced to the inversion process. The first is through prior information, while the second is through fixing a parameter at a certain value. In the former case, the second entry on the pertinent line of the parameter fix file should be "prior_info"; in the latter case the entry should be "fix_param".

The third entry on each line of the parameter fix file must contain the preferred value for the parameter whose name leads the line. This is the value at which the parameter will be fixed (for the "fix_param" option), or the "preferred value" assigned to the parameter through a prior information equation (for the "prior_info" option). Note that PARAMFIX checks that the value assigned to the parameter in this manner is between its upper and lower bounds as recorded in the PEST control file. If this is not the case, PARAMFIX will cease execution with an appropriate error message.

A fixed parameter cannot feature in any prior information. If an existing PEST control file contains one or more prior information equations which include a parameter that is to be fixed, those equations must be modified. Two options exist for modifying such an equation. The first is simply to remove the equation from the PEST control file. The second is to remove only the terms of the prior information equation that pertain to newly-fixed parameters; the values of those terms are then subtracted from the right hand side of the prior information equation after substituting parameter values read from the parameter fix file. The first of these options is implemented if the fourth entry on the pertinent line of the parameter fix file is "remove_prior". The second option is implemented if the fourth entry on the pertinent line of the parameter fix file is "retain_prior".

If outside parameter information is introduced to the inversion process through the addition of new prior information equations (i.e. if the second entry on the pertinent line of the parameter fix file is "prior_info"), then a little extra information must be supplied. As has already been mentioned, the preferred value for the parameter must be supplied as the third item on the line. The fourth item on the line must be "log" or "none". If it is supplied as "log", the prior information equation written by PARAMFIX for that parameter will actually pertain to the log of the parameter rather than to the parameter itself. However PARAMFIX will only allow this if the parameter is already logarithmically transformed in the parameter estimation process as designated by an appropriate PARTRANS value for that parameter in the "parameter data" section of the PEST control file. Similarly, if the fourth entry on the pertinent line of the parameter fix file is supplied as "none", indicating that the new prior

information equation is to pertain to the native parameter rather than to the log-transformed parameter, this will only be allowed if the parameter is designated as untransformed in the parameter estimation process (PARTRANS value of "none"). Note that the parameter value supplied as the third entry on the pertinent line of the parameter fix file (the parameter's preferred value) is log-transformed prior to being used in a prior information equation if the fourth entry is "log".

The fifth and sixth entries on any line of a parameter fix file which pertains to a new prior information equation must contain the weight to be assigned to the new prior information equation and the observation group to which the new equation should be assigned. The weight can be zero or positive. The observation group name may pertain to a group which already exists in the PEST control file to be modified by PARAMFIX, or it may pertain to a new observation group. In the latter case PARAMFIX will add the name of the group to the "observation groups" section of the PEST control file and increment the value of the NOBSGP variable accordingly.

4.6.3 Running PARAMFIX

PARAMFIX is run using the command

```
paramfix fixfile pestfile1 pestfile2
```

where

fixfile	is the name of a parameter fix file,
pestfile1	is the name of an existing PEST control file, and
pestfile2	is the name of the PEST control file to be written by PARAMFIX.

As mentioned above, it is important that the integrity of both the input and the output PEST control files are checked using PESTCHEK.

4.7 ADDPRED1

ADDPRED1 writes a PEST control file which instructs PEST to run in "predictive analysis" mode. It also adds prior information to a PEST control file. It is anticipated that this utility will be used most often when undertaking data space inversion. Hence it is documented in the section of this manual which deals with this subject. It can, however, be used in other circumstances as well.

4.8 ADDREG1

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 this 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 comprise a Tikhonov regularisation scheme. ADDREG1 assigns each equation to a regularisation group whose name begins with "regul_". ADDREG1 provides the second part of this name with the first six letters of the name of the parameter group to which the

pertinent parameter belongs. With parameter regularisation prior information equations thus assigned to different regularisation groups, the IREGADJ variable in the "regularisation" section of the PEST control file is 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, you should alter the groups to which parameters are assigned in the original PEST control file and re-run ADDREG1, or 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.

ADDREG1 is run using the command

```
addreg1 case1 case2
```

where

case1	is the filename base or full name of an existing PEST control file, and
case2	is the filename base or full name of the PEST control file which ADDREG1 must write.

Regardless of the PESTMODE setting of the first PEST control file, ADDREG1 sets the PESTMODE control variable to "regularisation" in the PEST control 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 1.0E-10 and PHIMACCEPT a value of 1.05E-10 in the new PEST control file; these may warrant alteration by the user. The FRACPHIM variable is assigned a value of 0.1 in the new PEST control file. Thus, during any iteration of the inversion process, the target objective function "seen" by PEST is 0.1 times its value at the beginning of that iteration. Also, as stated above, ADDREG1 assigns IREGADJ a value of 1.

It is very good practice to restrict the names of parameter groups in an existing PEST control file to 6 characters or less (even though 12 characters are allowed). When ADDREG1 forms a regularisation group name by prefixing a parameter group name with the text string "regul_", characters in the parameter group name after the sixth are lost. If two or more parameter group names are greater than six characters in length but have the first six characters in common, then the observation group names assigned to prior information equations which cite parameters from these different parameter groups will be the same. As well as this, there will then be duplication of observation group names in the "observation groups" section of the PEST control file as ADDREG1 adds these new observation group names to existing observation group names. PESTCHEK will detect this error so that you will then have the opportunity to rename observation groups and re-assign prior information equations to re-named groups accordingly. However keeping parameter group names to 6 characters or less in length will forestall the occurrence of this error altogether.

4.9 ADDREG2

ADDREG2 is similar to ADDREG1. However, unlike ADDREG1, ADDREG2 requires that the user provide a value for the target measurement objective function (i.e. PEST variable PHIMLIM) for inclusion in the "regularisation" section of the new PEST control file.

PHIMACCEPT is automatically set to 2 percent higher than this. Optionally, ADDREG2 allows a user to supply values for the FRACPHIM, IREGADJ, REGCONTINUE and UPTESTMIN control variables.

Another difference between ADDREG2 and ADDREG1 is that ADDREG2 insists that the PEST control file which it reads has PESTMODE set to "estimation". Of course, PESTMODE is set to "regularisation" in the PEST control file which it writes.

ADDREG2 is run using the command

```
addreg2 case1 case2 phimlim [fracphim] [iregadj] [continue] [hp]
```

where

case1	is the filename base or full name of an existing PEST control file,
case2	is the filename base or full name of the PEST control file which ADDREG2 must write,
phimlim	is the value for the target measurement objective function,
fracphim	(optional unless a value is supplied for IREGADJ) is the value for the FRACPHIM regularisation control variable,
iregadj	is the value for the IREGADJ regularisation control variable,
"continue"	(an optional variable) sets the REGCONTINUE flag to "continue" in the "regularisation" section of the new PEST control file, and
"hp"	(an optional variable) sets the UPTESTMIN control variable to 20 in the "control data" section of the new PEST control file.

If a value is not supplied for FRACPHIM, ADDREG2 sets it to 0.05. If a value is not supplied for IREGADJ, ADDREG2 provides its own value of 1. If a value of 4 is provided for IREGADJ through the command line, then ADDREG2 sets the NOPTREGADJ and REGWEIGHTRAT regularisation control variables to 1 and 20 respectively. If a value of 5 is provided for IREGADJ through the command line, then ADDREG2 sets the NOPTREGADJ, REGWEIGHTRAT and REGSINGTHRESH regularisation control variables to 1, 20 and 1.0E-5 respectively.

If used, the word "continue" can be placed anywhere on the command line. It should NOT be enclosed in quotes.

If used, the word "hp" can be placed anywhere on the command line. It should NOT be enclosed in quotes. This instructs ADDREG2 to set the UPTESTMIN control variable to 20. This variable is used by PEST_HP. It instructs PEST_HP to devote at least 20 model runs to lambda-testing during each iteration of the inversion process. This ensures that maximum use is made of the Jacobian matrix when trying to establish a parameter upgrade direction. Note, however, that ADDREG2 will not alter an UPTESTMIN value that is already supplied in the *case1* input PEST control file. Nor will it provide a value for UPTESTMIN if a value for UPTESTLIM is provided in the *case1* input PEST control file.

Note also that, in writing the new PEST control file, ADDREG2 sets JACUPDATE to 999. Thus it instructs PEST to employ Broyden Jacobian updating. However it will not alter the value of a JACUPDATE variable that is already supplied in the *case1* input PEST control file.

4.10 ADDREG2A

As the name implies, program ADDREG2A performs an almost identical role to program ADDREG2. However it uses the "regulgroup" keyword to identify regularisation groups in the "observation groups" section of the PEST control file that it writes. This avoids the need to append a "regul_" prefix to the names of parameter groups to form the names of regularisation groups when adding parameter-specific prior information equations to the "prior information" section of the new PEST control file. Regularisation group names can therefore be 12 characters in length; their names are inherited from parameter group names without the need to shorten these names.

4.11 ADDREG3

The task of ADDREG3 is similar to that of ADDREG1. However instead of adding prior information equations to a PEST control file, ADDREG3 adds observations. An observation is added for every adjustable parameter. The "observed value" of each observation is zero. The actual value is calculated as the difference between the parameter's current value, and its initial value as recorded in the PEST control file (or the log of the parameter's current value and the log of its initial value, if the parameter is log-transformed). The differencing calculation is undertaken by PAR2PAR. Hence the command to run PAR2PAR must be added to the model batch or script file which is run by PEST.

ADDREG3 is run using the command:

```
addreg3 case1 case2 weight areg
```

where

case1	is the filename base or full name of an existing PEST control file,
case2	is the new PEST control file written by ADDREG3,
weight	is the weight to assign to all new observations, and
areg	must be supplied as "yes" or "no".

The existing PEST control file must instruct PEST to run in "estimation" mode. It must contain no prior information. The new PEST control file contains a set of new observations – one for each adjustable parameter. Each new observation is named after the parameter to which it pertains. The observation name is obtained by adding the prefix "*pv_*" to the pertinent parameter's name. ("pv" stands for "preferred value.) The observation group to which it belongs is obtained by prefixing "*pv_*" to the name of the parameter group to which the parameter belongs if *areg* is supplied as "no", or by prefixing "*regul_*" to the name of the parameter group if *areg* is supplied as "yes". In both cases the parameter group name is truncated to 12 characters if this is necessary. The weight assigned to all new observations is equal to that provided on the ADDREG3 command line. A user may need to adjust this weight for some observations by direct editing of the PEST control file (especially if PEST is not run in "regularisation" mode and/or if the regularisation control variable IREGADJ is set to 0 or omitted from this file). If *areg* is supplied as "yes", ADDREG adds a "regularisation" section to the new PEST control file (in which IREGADJ is set to 1), and sets the PESTMODE control variable to "regularisation".

As was stated above, parameter-to-preferred-value differencing is undertaken by the PAR2PAR utility. ADDREG3 adds the name of a PAR2PAR input file and complementary template file to the "model input/output" section of the new PEST control file which it writes.

The names of these files are *par2par__.in* and *par2par__.tpl*. (Use of these unusual filenames lowers the possibility that ADDREG3 will overwrite an existing file of the same name that is used for another purpose.) ADDREG3 also adds the name of the PAR2PAR output file and the complementary instruction file that PEST must use to read this file to the "model input/output" section of the PEST control file. These files are named *par2par__.out* and *par2par__.ins*. Meanwhile the name of the template file that is used by PAR2PAR to write its output file is *par2par__out.tpl*. ADDREG3 writes all of *par2par__.in*, *par2par__.tpl*, *par2par__.out* and *par2par__.ins*.

In order for PAR2PAR to calculate differences between parameters and their preferred values (or between the logs of parameters and the logs of their preferred values), the following command must be added to the batch or script file which PEST runs as the model.

```
par2par par2par__.in
```

ADDREG3 reminds you of this before it ceases execution.

ADDREG3 can be useful in preparing for the running of RSI_HP. This program cannot read a PEST control file which contains prior information. Its task, after all, is to calculate sets of parameters that are as different as they can be, yet can still calibrate a model. Sometimes, however, it is useful to employ RSI_HP purely for the purpose of history-matching, and to encourage the ensemble to collapse in search of parameter uniqueness. This makes use of RSI_HP's ability to match a calibration dataset well with a minimal number of model runs. This can be achieved by seeking parameter sets that deviate minimally from their initial values, while using differences in these parameter sets as a basis for finite-difference derivatives calculation.

4.12 ADDCOVMAT

Observation groups that are created for regularisation purposes by ADDREG1 and ADDREG2 are not ascribed any covariance matrices. There are many inversion contexts, especially those where parameters have spatial connotations, where a covariance matrix is required. If parameters pertain to pilot points, covariance matrices can be built using programs such as PPCOV, PPCOV3D, PPCOV_SVA and PPCOV3D_SVA from the PEST Groundwater Data Utilities suite. The names of the covariance matrices that are produced by programs such as these can be added manually to PEST control files written by ADDREG1 and ADDREG2. Alternatively, this process can be automated using ADDCOVMAT. Automation can be of assistance where it must be repeated many times, possibly as part of a batch process that attempts to exert calibration constraints on a multiplicity of parameter fields.

ADDCOVMAT is run using the command

```
addcovmat case1 covmatfile case2
```

where

case1 is the filename base or full name of an existing PEST control file,
covmatfile is a file which links the name of a covariance matrix file to an observation group, and
case2 is the new PEST control file written by ADDCOVMAT.

An example of a *covmatfile* file is provided in Figure 4.4.

```
# observation_group_name  covariance_matrix_file
  k1x_regul               k1x_cov.mat
  k2x_regul               k2x_cov.mat
```

Figure 4.4 An example of an ADDCOVMAT input file which links covariance matrix filenames to observation group names.

Any line beginning with the "#" character is treated as a comment. The two entries on each non-comment line must be space-delimited, with the name of an observation group on the left and the name of a covariance matrix file on the right. The latter must be surrounded by quotes if it contains a space.

Note that, as is discussed above, ADDREG1 and ADDREG2 provide names for observation groups used for regularisation purposes themselves. These names are based on the names of parameter groups; a "regul" suffix is included in the name. ADDREG2A also provides names for observation groups. These are the names of respective parameter groups; a "regul_" prefix is not needed because of its use of the "regulgroup" convention to identify regularisation groups.

4.13 SUBREG1

As the name suggests, SUBREG1 performs the opposite task to that of ADDREG1; it subtracts regularisation from a PEST control file. It is run using the command

```
subreg1 case1 case2
```

where

case1 is the filename base or full name of an existing PEST control file, and
case2 is the filename base or full name of a new PEST control file.

SUBREG1 undertakes the following tasks when writing the new PEST control file.

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" or are identified as regularisation groups through use of the "regulgroup" string; the value of the NOBSGP variable in the "control data" section of the new PEST control file is reduced 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" or are identified as regularisation groups through use of the "regulgroup" string in the "observation groups" section of the PEST control file; the value of the NOBS variable in the "control data" section of the new PEST control file is reduced 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" or are identified as regularisation groups through use of the "regulgroup" string in the "observation groups" section of the PEST control file; the value of the NPRIOR variable in the "control data" section is reduced 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 on the new PEST control file until PESTCHEK gives you the all-clear.

4.14 SIMCASE

SIMCASE stands for "simplify case". It reads a PEST control file and a corresponding Jacobian matrix file (i.e. JCO 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 observations whose weights are 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 retaining the names of the original 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 and members of the PREDVAR and PREDUNC suites for parameter/predictive error and uncertainty analysis. The fact that regularisation data has been removed from the PEST input dataset facilitates use of these utility programs.

SIMCASE is run using the command

```
simcase case1 case2
```

where

case1 is the filename base or full name of an existing PEST control file, and
case2 is the filename base or full name of a new PEST control file.

The integrity of a SIMCASE-produced PEST control file can be checked with the PESTCHEK utility; however PESTCHEK must be run with the "/s" switch. Correspondence between the newly-created PEST control and Jacobian matrix files can be verified using the JCOCHEK utility.

4.15 WTFACTOR

WTFACTOR reads one PEST control file and writes another. In doing this it multiplies all weights pertaining to a user-nominated observation group by a user-supplied factor. As is discussed in part I of this manual, both observations and prior information equations should be assigned to one or a number of observation groups. WTFACTOR carries out weights multiplication irrespective of whether items belonging to the user-specified observation group are observations or prior information equations.

WTFACTOR is run using the command

```
wtfactor pestfile1 obsgroup factor pestfile2
```

where

pestfile1 is the name of an existing PEST control file,

obsgroup is the name of an observation group cited in that file,
factor is the weights multiplier for this group, and
pestfile2 is the name of the new PEST control file written by WTFACTOR.

For example, to write a new PEST control file named *file2.pst* in which weights assigned to the observation group *regul* in the PEST control file *file1.pst* are multiplied by a factor of 1.0345, WTFACTOR should be run using the command

```
wtfactor file1.pst regul 1.0345 file2.pst
```

WTFACTOR carries out only minimal checking of the PEST control file which it reads. Thus it will overlook many types of errors or inconsistencies that may be present in this file, transferring these directly to the PEST control file which it writes. As always, the latter should be checked using PESTCHEK before being used by PEST. It is also a good idea to check the input PEST control file prior to running WTFACTOR, for WTFACTOR's behaviour can be unpredictable if this file is internally inconsistent or incorrect.

WTFACTOR is general in its application, being useable with PEST control files pertinent to all modes of PEST's operation. However there is one situation in which it will not accomplish its goal of weight multiplication, and will instead cease execution after writing an appropriate message to the screen. This occurs where a prior information equation belonging to the user-specified observation group is spread over two or more lines of the PEST control file and the prior information weight and observation group name are not on the same line. An example of such a prior information equation is shown below.

```
pi1 1.3 * log(ro1) + 3.2 * log(ro2) = 3.234 1.000
& obsgroup1
```

However WTFACTOR will have no problems if the equation is written as follows.

```
pi1 1.3 * log(ro1) + 3.2 * log(ro2) = 3.234
& 1.000 obsgroup1
```

4.16 PWTADJ1

4.16.1 General

Prior to estimating parameters using PEST, you must decide what weights should be assigned to different observations. In some inversion situations it is wise to assign weights such that they are all proportional to the inverse of the standard deviation of measurement noise associated with measurements featured in the calibration dataset. In other cases, however, weighting strategies should be adopted which accommodate the "structural" nature of model-to-measurement misfit born of model defects and imperfections of the model as a simulator of a real world system. This matter is extensively discussed by Doherty (2015).

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 inversion process, and at the beginning of every iteration of the inversion process as it progresses. One practical means through which relative weighting of different observation groups can be established in such a way as to accommodate the imperfect nature of models as simulators of real-world systems is to undertake the following procedure.

1. Assign observations to different groups on the basis of their differential information content and/or their different measurement types. For example in a groundwater model, borehole heads in layer 1 can be assigned to a different observation group

from borehole heads in layer 2, while observed head differences between these layers can be assigned to a third observation group.

2. Build a PEST input dataset in which correct within-group weighting has been set for all observation groups, but for which the weighting strategy between groups is yet to be determined.
3. Run PEST with the NOPTMAX control variable set to zero. PEST will run the model just once and print to the screen, and to its run record file, the contribution made to the objective function by each observation group.
4. 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 is 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. The information contained in each observation group will thus, hopefully, be equally "visible" to PEST.

PWTADJ1 was written to automate this weights adjustment strategy.

4.16.2 Running PWTADJ1

PWTADJ1 is run using the command

```
pwtadj1 case1 case2 contribution
```

where

case1	is the filename base or name of an existing PEST control file,
case2	is the filename name or 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. PWTADJ1 reads files *case1.pst* and *case1.rec*, assuming 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. 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.

3. 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, it will not adjust weights for any group whose name begins with "regul" or is identified as a regularisation group through use of the "regulgroup" string in the "observation groups" section of the PEST control file.
4. If a covariance matrix is assigned to a non-regularisation observation group, PWTADJ1 will not adjust weights for this group, as weights supplied in the PEST control file are not employed for this group; instead PEST calculates a weighting matrix for this group from the covariance matrix provided for the group. Because PWTADJ1 leaves this matrix untouched, it is the user's responsibility to multiply all elements of this matrix by a pertinent factor if objective function equalisation is to take place; this can be achieved using the MATSMUL utility documented elsewhere in this manual. To help you in this endeavour, PWTADJ1 writes the required adjustment factor to the screen for each affected observation group.
5. PWTADJ1 adjusts weights for both observations and (non-regularisation) prior information.
6. 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.)
7. Before running PWTADJ1, you should check the *case1* PEST input dataset using PESTCHEK.
8. 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 original 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.

4.17 PWTADJ2

4.17.1 General

PWTADJ2 accomplishes post-calibration observation weights adjustment. The PEST control file written by PWTADJ2 can then be used in conjunction with linear analysis utilities such as GENLINPRED (and programs run by it) described elsewhere in this manual.

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 endow each observation with a weight that is the inverse of the standard deviation of noise associated with the corresponding measurement. On the

assumption that the nominated PEST control file contains optimised parameter values (this can be constructed using the PARREP utility if desired), the expected value of the objective function corresponding to the first model run should then be roughly equal to the number of non-zero-weighted observations comprising the calibration dataset. Similarly, the contribution to the objective function made by each observation group should then be roughly 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 in representing the statistics of measurement noise, the contribution made to the objective function by the observation group should be roughly 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 when run in "estimation" mode, then the expected value of the objective function is $n–m$, where n is the number of non-zero-weighted observations and m 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, and observation-group-specific contributions to this objective function, achieve their desired values.

In achieving a target objective function of n or n-m, 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 $n_g(n-m)/n$; in the latter case the total objective function is n-m. Group-specific post-calibration weights adjustment may be employed in order to accommodate the fact that the parameter estimation process may encounter much more difficulty in fitting some observation types to field measurements than others. If this is taken as an indication of a greater-than-anticipated level of (measurement or structural) 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-m (rather than simply n) as the expected value of the total objective function (though this matters little if m is small).

A further problem with the n-m concept arises where regularised inversion is undertaken and the number of estimated parameters is high (as it often is when an inverse 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 well-posed parameter estimation problem. Where singular value decomposition is employed as a regularisation device, the dimensionality of the calibration solution space is equal to the number of pre-truncation 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. The "reference variance" or "standard error of weighted residuals" therefore becomes equal to 1.0. This can be a useful exercise to perform prior to running linear analysis utilities such as GENLINPRED and members of the PREDUNC and PREDVAR suites, or in building a PEST control file that is cited in an observation uncertainty file (see section 2.5 of this manual).

4.17.2 Using PWTADJ2

PWTADJ2 is run using the command

```
pwtadj2 case1 case2 use_groups [parameter_correction]
```

where

case1	is the filename base or name of an existing PEST control file,
case2	is the filename base or name of a new PEST control file,
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 *m* of the *n–m* term; if this item is omitted from the command line, *m* is assumed to be zero.

Upon commencement of execution, PWTADJ2 reads the nominated PEST control file. Then it reads the run record file associated with this PEST control file. It is thus assumed that PEST has been run using the 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 or -2; in the former case only one model run is carried out by PEST while in the latter case a Jacobian matrix is also calculated. Furthermore, as suggested above, on many occasions the PARREP utility will have been employed to install optimised parameter values as initial parameter values in the "parameter data" section of this PEST control file.

As discussed above, PWTADJ2 has two options in performing observation weights adjustment. If the "ng" option is chosen for the *use_groups* control variable, then the adjustment factor by which PWTADJ2 multiplies all observation weights is the same for all observation groups. However if the "g" option is chosen, then different adjustment factors are computed for different groups. In both cases the total objective function after weights adjustment will be equal to *n–m*, where *m* is selected through the final variable on the command line; if this variable is omitted, *m* 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 $n_g(n-m)/n$ 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 PEST matrix file format (see section 2.4 of this manual), 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 for observations that belong to any observation group whose name begins with "regul" or is identified as a regularisation group through use of the "regulgroup" string in the "observation groups" section of the PEST control file. 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", regardless 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. Regardless 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.)

4.18 PSTCLEAN

From version 15.0 onwards, PEST, BEOPEST, PEST_HP and all commonly-used PEST utilities accommodate the presence of PEST++ input data in a PEST control file. As is explained in PEST++ documentation, control variables that are used by PEST++ are supplied in keyword format on lines that are embedded in a traditional PEST control file. These lines are easily identified, as they begin with the characters "++"; they can be placed anywhere within the PEST control file.

A PEST control file can also include comments. These can be placed anywhere on any line of this file following the # character. If desired, PEST control variables can precede the # character which marks an ensuing comment.

Ambiguity can arise if a filename that is featured in a PEST control file includes the "#" character. PEST-suite programs, including PSTCLEAN, avoid confusion by treating the "#" character as the precursor to a comment only if it occurs at the start of a line, or is preceded by a space and is not surrounded by matching quotes. Hence a # character that occurs within the name of a template, instruction or covariance matrix file or in the name of a model command, will not be interpreted as the start of a comment. (If the "#" character is, indeed, part of the name of a file and is preceded by a space, then presumably the filename will be surrounded by quotes.)

PSTCLEAN is run using the following command.

```
pstclean case1 case2
```

where

case1 is the filename base or name of an existing PEST control file, and
case2 is the filename base or name of a new PEST control file.

Note that if only the filename base of a PEST control file is supplied on the PSTCLEAN command line, then an extension of "*.pst*" is assumed.

Note also that PSTCLEAN removes control variables that are specific to PEST_HP from a PEST control file. In particular, if it encounters any of the following variables in the "control data" section of a PEST control file, it removes them:

- ORR_NOT_FIRST
- UPTESTLIM
- RUN_SLOW_FAC
- RUN_ABANDON_FAC
- WIN_MRUN_HOURS
- ZEROSENVAL
- SOFTSTOPHOURS
- HARDSTOPHOURS
- RRFSAVE

5. JCO File Construction and Manipulation

5.1 Introduction

As is described in part I of this manual, PEST records a Jacobian matrix (i.e. a sensitivity matrix) in a Jacobian matrix file whose filename base is the same as that of the PEST control file and whose extension is "*.jco*". This file (commonly referred to as a "JCO file" herein) has many uses. It can be used simply for acquiring knowledge of what model outputs are sensitive to what parameters. More commonly it provides a basis for calculation of post-calibration statistics, for undertaking post-calibration linear error and uncertainty analysis, for setup of SVD-assisted inversion, and for implementation of null space Monte Carlo uncertainty analysis.

Purposes for which utilities described in the present chapter are used include the following.

- Viewing a Jacobian matrix (JACWRIT and JCO2MAT);
- Transferring a JCO file between WINDOWS and UNIX platforms (JCO2MAT and MAT2JCO);
- Construction of a JCO file from its parts (JCOPCAT and JCOORDER);
- Matching a JCO file to a PEST control file when the latter is altered (JCO2JCO);
- Extracting parts of a Jacobian matrix for use by other programs (e.g. JCO2VEC);
- Creating a Jacobian matrix file for linear functions of model outputs (JCODIFF and JCOCOMB);
- Creating a weighted Jacobian matrix file (WTSENOUT).

As usual, the order in which programs are described in this chapter reflects similarity of function; programs that perform similar roles, or complementary roles, are documented in proximity to each other.

5.2 JACWRIT

JACWRIT rewrites a binary JCO file in ASCII (i.e. text) format. It is run using the command

```
jacwrit jcofile textfile
```

where

jcofile	is the name of a binary Jacobian matrix file written by PEST, and
textfile	is the name of a text file to which JACWRIT should write the Jacobian matrix in a form which is fit for human consumption.

Note the following.

- Parameter and observation names are listed in the text file written by JACWRIT; hence each sensitivity (i.e. partial derivative) that is recorded in this file can be linked to a particular parameter/observation pair.
- Only adjustable parameters are represented in the file written by JACWRIT; fixed and tied parameters are not represented because they are not represented in the JCO file.
- The sensitivity of a parameter to which another parameter is tied reflects the fact that this parameter "carries" at least one other parameter through the inversion process.
- Derivatives reflect the transformation status of a parameter. Thus if a parameter is

log-transformed, the derivative with respect to the log of that parameter is stored in the JCO file and recorded in ASCII format by JACWRIT.

- Rows of the Jacobian matrix are wrapped into successive lines in the JACWRIT-produced text version of this matrix.

5.3 JCO2MAT

JCO2MAT reads a PEST-produced Jacobian matrix file. It re-writes the matrix contained therein in PEST matrix file format; this format is described in section 2.4 of this document. The Jacobian matrix is then amenable to processing using matrix manipulation utilities described elsewhere in this manual.

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.

JCO2MAT can be useful in transferring a binary JCO file between a UNIX platform and a WINDOWS platform. (A binary file produced by software that is run on one of these platforms is often unreadable by software run on the other.) First convert the JCO file to matrix format using JCO2MAT. Next transfer the resulting ASCII file to the other platform (possibly running the DOS2UNIX or UNIX2DOS utility as appropriate after making the transfer). Then use the MAT2JCO utility on the target platform to write a binary JCO file for that platform.

5.4 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 JCO file.

MAT2JCO is run using the command

```
mat2jco matfile jcofile
```

where

matfile is the name of a matrix file, and
jcofile is the name of a new JCO file whose task it is for MAT2JCO to write.

5.5 JROW2MAT

JROW2MAT extracts a row of the Jacobian matrix from a JCO file and writes that row as a $1 \times m$ matrix in PEST matrix file format, where m is the number of (adjustable) parameters featured in the JCO file.

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×*m* matrix is to be written.

In the matrix file written by JROW2MAT, the row name given to the single matrix row is the name of the observation or prior information equation pertaining to the extracted row of the Jacobian matrix. Meanwhile recorded matrix column names are the names of adjustable parameters featured in the Jacobian matrix file. See section 2.4 of this manual for specifications of a PEST matrix file.

5.6 JROW2VEC

JROW2VEC performs the same function as JROW2MAT followed by MATTRANS (see elsewhere in this manual for documentation of the MATTRANS utility). That is, it extracts a user-nominated row from a Jacobian matrix housed in a JCO file. However instead of writing the extracted row in the form of a row matrix, it writes it as a column matrix. Recall that a matrix with one column is in fact a vector.

Many of the linear error and uncertainty analysis utility programs documented elsewhere in this manual require that predictive sensitivities be supplied as a vector; this is the **y** vector featured in many of the equations describing linear analysis presented by Doherty (2015). Predictive sensitivities can be calculated using PEST if, in the pertinent PEST control file, predictions are presented to PEST as "observations". If the PEST control file also contains real observations comprising a calibration dataset, then these predictions should be given a weight of zero; the "observed value" of predictions is therefore immaterial. Alternatively, if a PEST run is dedicated solely to evaluation of predictive sensitivities, and no members of a calibration dataset are represented in this PEST control file, weights and "observed values" for all "observations" featured in the "observation data" section of the PEST control file are of no relevance. In this case PEST can be run with NOPTMAX set to -2 simply for the purpose of evaluating predictive sensitivities.

Matrix utility programs such as MATQUAD can also use predictive sensitivities extracted from a JCO file by the JROW2VEC utility; if used appropriately, MATQUAD can be used to evaluate 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 extracted row of the Jacobian matrix is to be written as a vector.

5.7 JCOL2VEC

JCOL2VEC is similar to JROW2VEC. However instead of extracting a row from the Jacobian matrix, it extracts a column. Numbers in this column list the sensitivities of all observations to a single parameter.

JCOL2VEC is run using the command

```
jcol2vec jcofile parname matfile
```

where

jcofile is the name of a Jacobian matrix file,
parname is the name of a parameter featured in that file, and
matfile is the name of the matrix file to which the extracted column of the Jacobian matrix is to be written as a vector.

5.8 JCO2JCO

Suppose that you have run PEST to calculate a Jacobian matrix. A JCO file then exists which complements the PEST control file. The Jacobian matrix may have been calculated for a number of reasons, these perhaps including preparation for SVD-assisted inversion, and/or preparation for linear analysis. Suppose now that you make some alterations to the PEST control file. These may include the following.

- fixing some parameters;
- tying some parameters to parent parameters;
- changing the SCALE and OFFSET of some parameters;
- removing some parameters from the PEST control file;
- adding or removing prior information;
- altering observation weights;
- adding or removing observation groups;
- adding or removing observations.

Suppose that you would now like to obtain a JCO file that complements the new PEST control file. There is no need to run PEST again, for a new JCO file corresponding to this new PEST control file can be calculated from the original JCO file corresponding to the original PEST control file using the JCO2JCO utility.

JCO2JCO reads an existing PEST control file and a corresponding JCO file. It then reads a second PEST control file, modified from the original PEST control file in some or all of the ways outlined above. It then calculates a Jacobian matrix and writes a corresponding JCO file for the second PEST control file.

JCO2JCO is run using the command

```
jco2jco case1 case2
```

where

case1 is the filename base of the PEST control file for which a JCO file exists, and
case2 is the filename base of a second PEST control file for which a JCO file is required.

The following should be noted.

1. A parameter cited in the first PEST control file does not need to be cited in the second PEST control file; however the reverse is not true.
2. An observation cited in the first PEST control file does not need to be cited in the second PEST control file; however the reverse is not true.
3. If a parameter is tied to another parameter in the first PEST control file, it must be tied to the same parameter in the second PEST control file (JCO2JCO cannot "unravel" the derivatives of tied parameters). However a parameter can be tied in the second PEST control file, but not in the first.

4. If a parameter is fixed in the first PEST control file it must be fixed in the second PEST control file. However a parameter can be fixed in the second PEST control file but not in the first.
5. The JCO file written by JCO2JCO does not cite any prior information present in either the first or second PEST control files. This should cause no problems for programs such as SVDAPREP or linear analysis utilities which use this file.
6. The first JCO file must have been calculated using version 8 or later of PEST. If this is not the case it can be translated to the newer JCO file format using the JCOTRANS utility.

JCO2JCO allows a parameter to have a different SCALE in the second PEST control file from that which it has in 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.

Normally JCO2JCO issues a warning message if the initial value of any parameter differs between the first and second PEST control files. However it will not issue a warning message if a parameter's initial value is different between these two PEST control files and the product of the parameter's initial value and its SCALE is the same in both files.

JCO2JCO's handling of parameter SCALE may be such as to disallow certain complex alterations to parameter status between the old and new PEST control files. For example JCO2JCO will object if a parameter is given a different SCALE, and is simultaneously assigned a different transformation or tied status between PEST control files. 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.

5.9 JCOTRANS

JCOTRANS translates a JCO file produced by version 7 or earlier of PEST to a JCO file compatible with version 8 or later of PEST. The latter file is recorded in more compressed form.

JCOTRANS is run using the command

```
jcotrans jcofile1 jcofile2
```

where

jcofile1 is the name of a Jacobian matrix file written in the old format, and
jcofile2 is the name of the file to which the Jacobian matrix will be recorded in the new format.

For both of these filenames, the "*.jco*" extension can be included or omitted; if it is omitted JCOTRANS appends it automatically.

5.10 JCOPCAT

JCOPCAT concatenates two Jacobian matrices contained in two different JCO files. Concatenation is carried out with respect to parameter values rather than observations; that is the Jacobian matrices contained in the respective JCO files are concatenated "sideways" so that extra parameter columns are added to the file. (Recall that each column of a Jacobian matrix file pertains to a single parameter and each row pertains to a single observation.)

JCOPCAT can be used to create a single JCO file from two JCO files created during two separate PEST runs. These runs must be based on different adjustable parameters but must feature the same observations. Use of JCOPCAT thus removes the need for re-calculation of sensitivities with respect to existing parameters if new parameters are added to a PEST control file, or if the status of these parameters changes from fixed to adjustable. It can also be used for calculation of partial Jacobian matrices on different machines prior to "stitching these matrices together" to form an entire Jacobian matrix.

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, it may be possible to prepare JCO files for concatenation using the JCOORDER utility. This may be required if, for example, prior information is featured in one JCO file but not in another. (Note that if JCO2JCO is then used to adapt a JCOPCAT-produced JCO file to another PEST control file, it ignores such prior information; thus prior information can be removed from one or both of the JCO files read by JCOPCAT before concatenation without any loss of information which may compromise further use of the resulting JCO file.)

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 then writes the new, concatenated JCO file.

5.11 JCOMIX

JCOMIX constructs a JCO file to complement a PEST control file from elements of existing JCO files. It is run using the following command

```
jcomix pestfile3 jcofile1 jcofile2 jcofile3
```

where

pestfile3	is an existing PEST control file,
jcofile1	is an existing Jacobian matrix file (referred to as a "primary" JCO file),
jcofile2	is another existing Jacobian matrix file (referred to as a "secondary" JCO

file); and

jcofile3 is a new PEST control file that complements *pestfile3*.

JCOMIX begins execution by reading the PEST control file *pestfile3*. It then reads the Jacobian matrix contained in file *jcofile1*. Wherever it finds an element of this matrix that has the same parameter and observation name as an element of the matrix intended for file *jcofile3*, it fills the latter matrix element with the value of the *jcofile1* matrix element.

Next JCOMIX reads the Jacobian matrix contained in file *jcofile2*. Where it finds elements with that matrix with parameter and observation names that correspond to elements of the Jacobian matrix intended for file *jcofile3*, it transfers them to the latter matrix, but only if they have not already been filled by element transfer from *jcofile1*.

Finally JCOMIX determines whether values have been assigned to all elements of the *jcofile3* matrix. If they have all been assigned a value, then JCOMIX records the corresponding Jacobian matrix in the *jcofile3* JCO file. If not, it ceases execution with an appropriate error message.

Note the following important aspects of JCOMIX usage:

- The PEST control file cited in the JCOMIX command line must have an extension of "*.pst*";
- JCO files cited on the JCOMIX command line must have extensions of "*.jco*";
- Normally, *pestfile3* and *jcofile3* will have the same filename base;
- It is the user's responsibility to ensure that parameters that are log-transformed in file *pestfile3* are also log-transformed in the PEST control files that complement JCO files *jcofile1* and *jcofile2*. JCOMIX has no way of checking this. If a user realizes that transformation states differ, he/she can build a Jacobian matrix for the correct parameter transformation states using the JCO2JCO utility.
- If file *pestfile3* contains prior information, the coefficients that occur within its prior information equations constitute pertinent elements of the *jcofile3* Jacobian matrix. JCOMIX does not over-write these elements with those of corresponding elements of matrices contained in either of files *jcofile1* and *jcofile2*. Nor does it worry if these prior information coefficients are missing from matrices contained in the latter two files; the *jcofile3* Jacobian matrix includes these coefficients regardless.

5.12 JCOORDER

JCOORDER reads a JCO file. It then writes another JCO file after performing one or more of the following tasks on the Jacobian matrix housed in the original 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 JCO file,
orderfile is a parameter/observation ordering file (see below) or PEST control file, and
jcofile2 is a new JCO file written by JCOORDER.

Figure 5.1 illustrates a parameter/observation ordering file.

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

Figure 5.1 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 read by JCOORDER. 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 rather than a parameter/observation ordering file through the extension "*.pst*" provided for 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.

If using a PEST control file instead of a parameter/observation ordering 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 parameters and/or observations which are not cited in the JCO file must not be present in the PEST control file.

Use of JCOORDER can complement the use of JCOPCAT in preparation for SVD-assisted inversion 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 before running SVDAPREP to build the super parameter PEST control file. If these files contain different items of prior information JCOORDER can be used to remove prior information from both of these JCO files prior to concatenation using JCOPCAT.

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 a base PEST control file which the concatenated JCO file is being built to complement prior to running SVDAPREP.

5.13 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 used by JCOORDER. Like JCOORDER, it 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"; 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. 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 JCO file,
addfile	is a parameter/observation addition file, and
jcofile2	is a new JCO file written by JCOADDZ.

Specifications for a parameter/observation addition file are identical to that of a parameter/observation ordering file; see figure 5.1.

Note the following aspects of JCOADDZ's operations.

1. It is not essential that both parameters and observations be added to an existing JCO file. Thus the parameter/observation addition file may cite no observations or no parameters. However, regardless of whether or not any parameters or observations are cited in each section of this file, the "* parameters" and "* observations" headers must still be featured in the file.
2. As for a parameter/observation ordering file, blank and comment lines are allowed in

a parameter/observation addition file.

Extra parameters and observations are appended to the last column and row respectively of the Jacobian matrix housed in the existing JCO file. If it is desired that they occupy different rows and/or columns from the last, this can be accomplished through use of the JCOORDER utility.

5.14 JCODIFF

JCODIFF subtracts the contents of one Jacobian matrix file from that of another. It can be useful when conducting linear analysis. It facilitates construction of a JCO file which can provide the basis for exploration of 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 JCO file,
jcofile2	is another existing JCO file, and
jcofile3	is a new JCO file, written by JCODIFF, containing a Jacobian matrix which is the difference between the Jacobian matrices housed in the above two JCO files.

Note the following.

1. The first and second JCO 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 JCO file is subtracted from that contained in the first in forming the Jacobian matrix written to the third JCO file.

5.15 JCOSUM

JCOSUM performs the weighted sum of two Jacobian matrices. It can be useful when updating a randomized Jacobian matrix calculated by RRF2JCO with a supplementary randomized Jacobian matrix calculated from the outcomes of new model runs; in this case the weights (i.e. the factors by which the contents of each Jacobian matrix are multiplied) should add to 1.0.

JCOSUM is run using the command

```
jcosum jcofile1 factor1 jcofile2 factor2 jcofile3
```

where

jcofile1	is an existing JCO file,
factor1	is the factor by which all elements of the Jacobian matrix contained in this file are multiplied,
jcofile2	is another existing JCO file,
factor2	is the factor by which all elements of the Jacobian matrix contained in this file are multiplied, and
jcofile3	is a new JCO file, written by JCOSUM.

The matrix contained in the new JCO file is equal to *factor1* times the first Jacobian matrix plus *factor2* times the second Jacobian matrix.

5.16 JCOCOMB

5.16.1 General

JCOCOMB reads one JCO file and writes another. Observations pertaining to the second JCO file are user-specified linear combinations of those recorded in the first JCO file. JCOCOMB combines sensitivities (i.e. elements of the Jacobian matrix) in the same proportions to produce a new Jacobian matrix pertaining to the combined observations.

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 PEST 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 PEST control file. 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.

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. Figure 5.2 illustrates such a file.

```
* 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
```

Figure 5.2 Example of 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 name 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 sensitivity of the composite observation to each such parameter. 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.

5.16.2 Using JCOCOMB

JCOCOMB is run using the command

```
jcocomb case1 case2 combfile
```

where

case1	is the filename base of the first PEST control file,
case2	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 parameter estimation, it is nevertheless useable by utilities such as those which comprise the PREDVAR and PREDUNC linear analysis 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-suite and PREDUNC-suite processing, where it is desired that predictive errors and uncertainties be 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-suite and PREDUNC-suite utilities by the JROW2VEC utility prior to PREDVAR-suite and PREDUNC-suite processing.

5.17 JCOSUB

JCOSUB can be useful if you suspect that derivatives with respect to some parameters in an existing JCO file need improvement. Improvement of these derivatives may be accomplished using the following procedure.

1. Create a new PEST control file in which all parameters but the worrisome ones are fixed. Employ settings in this new PEST control file which hopefully promulgate better derivatives of model outputs with respect to these parameters (e.g. use a three or five point finite-difference stencil).
2. Run PEST using this new PEST control file with NOPTMAX set to -2 to create a corresponding JCO file.
3. Run JCOSUB to substitute elements in the old JCO file with elements from the new JCO file.

JCOSUB is run using the command

```
jcosub jcofile1 jcofile2 jcofile3
```

where

jcofile1 is the name of an existing JCO file,
jcofile2 is the name of another existing JCO file, and
jcofile3 is a new JCO 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 first, then the respective element of the first is overwritten by that of the second.

In order to save confusion, 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 by those in the second.

5.18 JCOZERO

JCOZERO allows a user to set individual elements, or groups of elements, of a Jaobian matrix to zero. The Jacobian matrix is housed in a JCO file. The altered Jacobian matrix is written to another JCO file.

JCOZERO is run using the following command.

```
jcozero jcofile1 zerofile jcofile2
```

where

jcofile1 is the name of an existing JCO file,
zerofile is a file which denotes which Jacobian matrix elements are to be zeroed, and

jcofile2 is the new JCO file to be written by JCOZERO.

Figure 5.3 shows a simple example of a file which JCOZERO reads in order to learn which Jacobian matrix elements it must set to zero.

```
# Example of a Jacobian zeroing file

* parameters
k_ppt1
k_ppt2
* observations
well01_1
well02_1

* parameters
k_ppt10*
storage?20
* observations
part*
part_time
```

Figure 5.3 A file used by JCOZERO for denoting which elements of a Jacobian matrix are to be zeroed.

The "Jacobian zeroing file" is divided into blocks. Each block is comprised of a "parameters" sub-block and an "observations" sub-block. The former sub-block must begin with the string "* parameters" whereas the latter sub-block must begin with the string "* observations". These sub-blocks must be provided in that order.

Each line of both of these sub-blocks must contain a single entry. For the parameters sub-block this must be the name of a parameter, or a string from which a group of parameter names can be constructed. In the latter case, "*" denotes one or more infilling characters while "?"denotes a single infilling character. The same applies to entries within the "observations" sub-block (except that individual observations or groups of observations are denoted instead of parameters). Note that parameter/observation names (or strings which identify a group of such names) must not be surrounded by quotes in ether of these sub-blocks.

Blank lines can appear anywhere within a Jacobian zeroing file. Any line beginning with the "#" character is treated as a comment and is thus ignored.

5.19 JCOBLANK

5.19.1 General

JCOBLANK writes a file that informs programs such as PEST_HP that certain elements of its Jacobian matrix must be assigned a value of zero. It thus provides a basis for user-prescribed "localization". This can upgrade the integrity of randomized Jacobian matrix calculation. It also supports Jacobian matrix calculation through finite parameter differencing in which more than one parameter is incremented during any particular model run. Simultaneous parameter incrementation becomes a possibility if the set of model outputs that are sensitive to one parameter has no commonality with the set of model outputs that are sensitive to another parameter. Optionally, JCOBLANK devises a model run strategy through which simultaneous parameter increments may be used to fill the Jacobian matrix. This run strategy is recorded in a "simultaneous increment file".

Internally, JCOBLANK builds and stores a "pseudo Jacobian matrix". This is a matrix whose elements are all logical variables, and whose values are thus either FALSE or TRUE; FALSE indicates a zero-valued Jacobian matrix element while TRUE indicates a non-zero-valued Jacobian matrix element. This pseudo Jacobian matrix has NOBS rows and NESPAR

columns, where NOBS is the number of observations featured in a PEST input dataset and NESPAR is the number of adjustable parameters featured in this dataset; these are read from a PEST control file. Note that JCOBLANK will not zero elements of a Jacobian matrix that pertain to prior information equations.

5.19.2 Zero File

JCOBLANK reads a file which contains the information which it requires to determine which elements of a Jacobian matrix must be zeroed. This can be an existing Jacobian matrix file (i.e. a JCO file); alternatively, it can be a so-called "zero file". Use of the former file is discussed below. The latter type of file is discussed in this subsection. Note that specifications of the zero file read by JCOBLANK differ from those of the zero file read by JCOZERO; they are more powerful. Figure 5.4 exemplifies a zero file read by JCOBLANK.

```
* start all nonzero

* zero parameter observation_group
  ro1         obsgp2
  ro2         obsgp2
  ro9         obsgp1
  ro10        obsgp1

* nonzero parameter observation
  ro2         ar9
```

Figure 5.4 A file used by JCOBLANK for denoting which elements of a Jacobian matrix must be zeroed.

Blank lines can appear anywhere in a zero file. Lines beginning with the "#" character are treated as comments; hence they are ignored. Section headers begin with the "*" character.

The first non-blank, non-comment line of a zero file must be:

```
* start all zero
```

or

```
* start all nonzero
```

In the former case all elements of the pseudo-Jacobian matrix filled by JCOBLANK are given the logical value of FALSE. In the second case they are all given the logical value of TRUE.

Each section header must be comprised of three entries. The first entry must be "zero" or "nonzero". In the former case, the section header instructs JCOBLANK to set certain elements of its pseudo Jacobian matrix to FALSE; these elements are identified in lines which follow the section header. In the latter case, respective elements of the pseudo Jacobian matrix are set to TRUE.

The next two entries of the section header indicate how to interpret the two columns that are recorded on subsequent lines of the section. The options for these entries are "parameter", "parameter_group", "observation" and "observation_group". Any of these can comprise the first entry. However if this first entry is "parameter" or "parameter_group", the second entry must be "observation" or "observation_group". Similarly, if the first entry is "observation" or "observation_group", the second entry must be "parameter" or "parameter_group".

Then follow two columns of names. These names must refer to entities indicated in the section header (i.e. parameters, parameter groups, observations or observation groups).

Individual elements or groups of elements of the pseudo Jacobian matrix are endowed with values of TRUE or FALSE in accordance with these entries.

To support flexibility in character string matching, the name of a parameter or observation (but not a parameter group or observation group) can include a single "*" or "?" character (but not both of these characters); however this is only allowed for names appearing in the second column of the table comprising each section. As is usual practice, in matching one character string to another, the "*" character can be replaced by a group of characters while the "?" character is replaced by a single character in forming the match.

5.19.3 Using a Jacobian Matrix for Zeroing

Instead of reading a zero file, JCOBLANK can read a Jacobian matrix file (i.e. a JCO file). If JCOBLANK is asked to read a Jacobian matrix file it asks the following questions:

```
Enter fraction-of-max for observation blanking:
Enter fraction-of-max for parameter blanking:
```

The number supplied in response to each of these questions must not be less than zero or greater than one. The first of the above variables will be referred to as FRACOBS while the second will be referred to as FRACPAR. JCOBLANK uses the following algorithm to zero elements of the Jacobian matrix:

- First it multiplies all rows of the Jacobian matrix by the weight associated with that row.
- For each row of the Jacobian matrix it sets elements to zero whose absolute value is less than FRACOBS times the highest absolute value occurring in that row.
- For each column of the Jacobian matrix, it sets elements to zero whose absolute value is less than FRACPAR times the highest absolute value occurring in that row.
- JCOBLANK then constructs the pseudo-Jacobian matrix by declaring non-zero-valued elements of the Jacobian matrix to be TRUE and zero-valued elements of the Jacobian matrix to be FALSE.

Note that JCOBLANK will terminate execution with an error message if the PEST control file that is associated with the Jacobian matrix file ascribes a covariance matrix to observation groups which do not pertain exclusively to prior information equations. This makes linkages of weights to rows of the Jacobian matrix difficult.

5.19.4 The JCO Blanking File and JCO Blanking Pattern File

JCOBLANK writes a binary "blanking file". This file must possess an extension of "*.jcz*"; it is read by PEST_HP if its name is cited in the "simultaneous parameter increments" section of a PEST_HP control file. Elements of this file are logical – TRUE for unblanked and FALSE for blanked. JCOBLANK also records this same information in a "localization JCO file". This is a traditional Jacobian matrix file. The elements of this file are real – 1.0 for unblanked and 0.0 for blanked. This file can be read by any PEST-suite program that reads a JCO file. It can also be used by PEST-HP for modification of a randomized Jacobian matrix if its name is cited in the "randomized Jacobian" section of a PEST_HP control file.

JCOBLANK also records the matrix on which JCO blanking is based. This is recorded in a "blanking pattern file". The full, logical, pseudo-Jacobian matrix computed by JCOBLANK is recorded in this ASCII file. In this matrix, "T" and "F" are used to represent TRUE and FALSE; recall that FALSE indicates a zero-valued Jacobian element.

5.19.5 The Simultaneous Increment File

If asked to do so, JCOBLANK records a "simultaneous increment file". This is an ASCII file containing two tables. The first table links each parameter with an increment counter, i.e. the index of a model run required for filling of the Jacobian matrix. The second table indicates what parameters are incremented during each model run. JCOBLANK calculates the total number of model runs required to fill the Jacobian matrix (on the assumption that forward – and not central – derivatives are employed to fill this matrix). This number is recorded on the screen and in its simultaneous increment output file. (If central derivatives are employed to fill this matrix then the number of model runs required to fill it is twice this amount.)

PEST_HP can be asked to read a simultaneous increment file to guide it in finite-difference derivatives calculation. If it uses this file, it must also use the corresponding blanking file written by JCOBLANK; this allows it to discriminate between changes in model outputs caused by simultaneous incrementation of multiple parameters.

Once a simultaneous increment strategy has been calculated on the basis of a blanking strategy, it may be found that the number of Jacobian elements that require blanking to avoid confusion in observation-parameter dependency can be reduced. In the extreme case, if the blanking structure of a Jacobian matrix makes any duplication of increments in any model run impossible, then there is no need to blank any Jacobian elements at all. In most cases, it will be possible to unblank some elements of the Jacobian matrix after a simultaneous increment strategy has been devised. When PEST_HP then uses the amended JCO blanking file in combination with the simultaneous increment file, some finite-difference calculated sensitivities may then have greater precision through avoidance of unrequired blanking.

5.19.6 Running JCOBLANK

JCOBLANK receives its instructions through user-supplied responses to a series of questions. It commences execution with the prompt:

```
Enter name of existing PEST control file:
```

The name of a PEST control file should be supplied in response to this prompt. This file may or may not include prior information. Furthermore, it may instruct PEST to run in any of its possible modes. The behaviour of JCOBLANK does not depend on any of these possibilities. It ignores prior information and writes a blanking file that pertains to parameters and observations only.

Its next prompt is:

```
Obtain blanking information from a zeroing file or a JCO file [z/j]:
```

If the response to the above prompt is "z" then JCOBLANK prompts for the name of a zero file. Alternatively, if the response is "j", JCOBLANK prompts:

```
Enter name of JCO file:
Enter fraction-of-max for observation blanking:
Enter fraction-of-max for parameter blanking:
```

JCOBLANK next asks for the name of the blanking file that it must write. The name supplied for this file must have an extension of "*.jcz*":

```
Enter name for blanking file:
```

JCOBLANK then asks for the name of a "localization JCO file". The prompt is:

```
Enter name for localization JCO file:
```

As stated above, this file can be used by PEST_HP for modification of its randomized Jacobian matrix. It is a binary Jacobian matrix file whose elements are either 0.0 (meaning blanked) or 1.0 (meaning unblanked). The extension provided for this filename must be "*.jco*" in accordance with conventions used for the naming of Jacobian matrix files.

Next JCOBLANK asks for the name of the blanking pattern file that it must write. The prompt is:

```
Enter name for blanking pattern file:
```

Respond with an appropriate name for this ASCII file.

JCOBLANK's next question is:

```
Write simultaneous increment file? [y/n]:
```

If the response to this prompt is "y", JCOBLANK asks for the name of the file that it must write. This, too, is an ASCII file. After receiving the name of this file, JCOBLANK asks:

```
Adjust blanking to conform with simultaneous increments? [y/n]:
```

to which a response of "y" or "n" should be supplied.

5.19.7 Observation Weights

Ideally, rows of a Jacobian for which observation weights are zero should be blanked. This is because these rows can be omitted from a Jacobian matrix with no adverse consequences for an inversion process. Hence the existence of any non-zero elements in these rows should not be allowed to compromise "blockiness" of a Jacobian matrix, and hence development of an efficient simultaneous increment strategy. JCOBLANK automatically blanks zero-weighted rows of a Jacobian matrix if it builds a blanking file based on sensitivities occurring within an existing Jacobian matrix. However it does not automatically blank these rows if it follows user-supplied blanking instructions provided in a zero file. It is thus the user's responsibility to ensure that these rows are blanked him/herself. There may be occasions when he/she does not wish that these rows be blanked. This may occur, for example, if an inversion process is "carrying" one or a number of model predictions for which linear uncertainty evaluation will later be undertaken.

5.20 JCOCHEK

This utility is documented in chapter 3 of this manual. It is used to test compatibility between a PEST control file and a Jacobian matrix after either has been manipulated or altered.

5.21 WTSENOUT

5.21.1 General

"WTSENOUT" stands for "weighted 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 matrix file and the ASCII residuals file respectively produced as an outcome of running PEST).
2. It calculates $\mathbf{Q}^{1/2}\mathbf{J}$ where $\mathbf{Q}$ is the weight matrix and $\mathbf{J}$ is the Jacobian matrix for the current problem. As is explained in part I of this manual, sensitivities comprising elements of the Jacobian matrix pertain only to adjustable parameters; the sensitivities

associated with each particular parameter reflect the transformation status of the parameter, and whether or not any other parameters are tied to it.

3. It computes $\mathbf{Q}^{1/2}\mathbf{o}$ where **o** is the model output vector corresponding to best-fit parameters.
4. It records $\mathbf{Q}^{1/2}\mathbf{J}$ in a binary JCO file (from which an ASCII version can be obtained using the JACWRIT or JCO2MAT utilities).
5. It writes the $\mathbf{Q}^{1/2}\mathbf{o}$ vector to an ASCII file in PEST matrix file format (see section 2.4 of this manual for specifications 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 which it is asked to read does not inform PEST to run in "estimation" mode. (If the previous PEST run employed another mode, it is a simple matter to create a new PEST control file in which PESTMODE is set to "estimation", and then use the PARREP utility to populate it with previously estimated parameters as initial parameters. The JCO2JCO utility can then be used to create a corresponding JCO file. PEST can then be run with NOPTMAX set to -1, and with the "/i" switch employed on the command line to compute a RES file pertaining to initial parameters; the existing Jacobian matrix is then read by PEST, this saving it the trouble of having to re-compute it.)
2. It is possible that the best set of parameters, and the model outputs corresponding to these recorded 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 correspond 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 you 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 to build the pertinent JCO and RES files.
3. WTSENOUT will accommodate the provision of covariance matrices for one or more observation groups in the PEST control file which it is asked to read.

5.21.2 Running WTSENOUT

WTSENOUT is run using the command

```
wtsenout pestfile matfile jcofile
```

where

pestfile	is the name of a PEST control file (a corresponding JCO and RES file must also exist),
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 weighted Jacobian matrix output file (i.e. the binary JCO file which will contain $\mathbf{Q}^{1/2}\mathbf{J}$).

PEST reads the nominated PEST control file and the corresponding JCO and RES files. It obtains the names of these latter two files by affixing pertinent extensions to the filename base of the PEST control file. (You should ensure that the new JCO file written by

WTSENOUT has a different name from that which WTSENOUT reads.)

5.22 JCOWT

JCOWT contains some of the functionality of WTSENOUT. It reads a PEST control file and a corresponding Jacobian matrix file. It writes a new Jacobian matrix file in which the elements of all rows of the Jacobian matrix are multiplied by weights associated with those rows. As presently programmed, JCOWT does not perform this task if a covariance matrix is associated with any observation group; under these circumstances it ceases execution with an appropriate error message.

JCOWT is run using the command

```
jcowt pestfile jcofile1 jcofile2
```

where

pestfile	is the name of a PEST control file,
jcofile1	is the name of an existing JCO file (normally the JCO file associated with the PEST control file), and
jcofile2	is a new JCO file written by JCOWT.

6. Integrity of Finite-Difference Derivatives

6.1 Introduction

With modern regularisation devices forming essential components of its parameter estimation algorithms, PEST's performance in highly-parameterized, ill-posed inversion contexts is generally good. While the time required for an inversion process to reach completion may be high if model run times are long, numerical stability of the inversion process is usually not a problem.

The most likely reason for problematical PEST behaviour is lack of integrity of finite-difference derivatives. This can be an outcome of model numerical granularity, so that small changes in the values of model outputs are not solely a function of small changes in the values of its parameters. Problematical finite-difference derivatives introduce numerical noise to the Jacobian matrix, thereby compromising PEST's parameter upgrade calculations. During an inversion process, noise in the Jacobian matrix may express itself through an objective function which falls during the early stages of the process, but which then abruptly ceases to fall any further, or may even erratically rise and fall over subsequent iterations.

As is discussed in part I of this manual, PEST provides a number of mechanisms which can ameliorate the effects of poor model performance on finite-difference derivatives. These include split slope analysis and the use of a five point finite-difference stencil. However if the numerical performance of a particular model is too bad, then even these measures will fail to prevent a severe deterioration in PEST's performance. In that case, it may not be possible to use PEST with that model at all. The use of proxy or surrogate models for derivatives calculation may then be contemplated. Alternatively, a global optimiser may then be used instead of PEST.

This chapter describes JACTEST and a supporting utility program which allows you to directly test the integrity of finite-difference derivatives. It also describes the MULJCOSEN utility which can provide some indications of problematical finite-difference derivatives. Assessing how bad finite-difference derivatives are likely to be allows you to develop an inversion strategy that can accommodate deficits in their integrity.

6.2 JACTEST

6.2.1 General

JACTEST is used to test the integrity of derivatives calculated by PEST. It runs the model a number of times with incrementally varied parameters. It monitors the same model outputs as those for which sensitivities with respect to the varied parameter must be calculated. By plotting the values of these outputs against the values of the varied parameter the presence of numerical noise can be easily detected.

6.2.2 Using JACTEST

JACTEST is run using the command

```
jactest case parname n outfile [/p]
```

where

case is the filename base 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 parameter is varied incrementally; this is parameter *parname* supplied through the JACTEST command line. 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 its output file (i.e. to the file *outfile* specified through the JACTEST command line). The first row of this table contains values of the nominated parameter employed for each model 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 derivatives 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 parameter nominated on the command line is tied or fixed.
3. Where the user-specified parameter has other parameters tied to it, the tied parameters are varied such that their ratio to the parent parameter remains unchanged; that is, they are varied in the same way that PEST varies them.
4. Parameter SCALEs and OFFSETs as provided in the "parameter data" section of the PEST control file are respected; their use is identical to that of PEST.
5. JACTEST will not transgress parameter bounds. If the addition of $n/2$ times the derivative increment to a given parameter value causes that parameter to exceed its upper bound, JACTEST will undertake more model runs with subtracted parameter increments in order to complete its n+1 runs. The opposite action is taken if a lower bound is transgressed. If both bounds are transgressed, JACTEST will cease execution with an appropriate error message.
6. In some circumstances the value of the user-specified parameter as recorded in the JACTEST output file may be slightly different from its expected incremented value. This is an outcome of the fact that, like PEST, JACTEST adjusts the value of a parameter slightly if the presence of a parameter space of limited width on one or more template files cited in the PEST control file requires that the parameter's value 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.)

6.2.3 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. The parallelisation protocol is the same as that of Parallel PEST; at the time of writing there is no BEOJACTEST. JACTEST thus 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*". See section 11.2 of part I of this manual for specifications of this file.

As stated, operation of JACTEST in parallel mode is the same as that of Parallel PEST. Agents (named PAGENT) must be started in working directories on other machines (or on the same machine if it has multiple processors). Communication between the manager program (in this case JACTEST) and its agents is through message files – the same message files as used by Parallel PEST. It is the agent, and not JACTEST, which runs the model in each case.

As for parallel PEST, agents should be started before PEST. However, like parallel PEST, JACTEST will tolerate the late arrival of agents.

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 agents. 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").

6.2.4 Stopping JACTEST

If the PSTOP or PSTOPST command is run in the JACTEST working directory (i.e. folder), 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 orphaned model runs from reaching completion in their agent windows. All agents will automatically shut down once their respective model runs are finished if the PSTOPST command is used to terminate JACTEST execution; however they will not shut down if the PSTOP command is employed. Meanwhile JACTEST writes all results that it has accumulated to the time of stoppage to its normal output file before it shuts itself down.

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 folder.

6.3 POSTJACTEST

As was described in the preceding subsection, 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.

However if the line is jagged, a potential problem in the finite-difference calculation of model 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 performs this role.

POSTJACTEST is run using the command

```
postjactest jactestfile thresh outfile
```

where

jactestfile	is the name of a JACTEST output file,
thresh	is a threshold value whose role is explained below which must be set to a number greater than zero, and
outfile	is the name of a file which POSTJACTEST writes.

POSTJACTEST reads the JACTEST output file line by line. Recall that values provided on each line of the JACTEST output file are the values of model-generated observations computed using incrementally-varied parameters. For any such line, POSTJACTEST computes differences between successive numbers on the line. 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 an indicator of parameter-dependent slope discrepancies. It is recorded on POSTJACTEST's output file, together with the observation name to which this measure pertains. While a high value for this number may reflect nothing more than model nonlinearity, it may also reflect contamination of finite-difference derivatives by numerical noise.

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 *thresh* 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 on the basis of its insensitivity to parameters. (Numerical granularity of these outputs would reflect truncation of all but the last couple of significant figures incurred by differencing rather than true model numerical granularity.)

The POSTJACTEST output file is easily imported into a spreadsheet such as EXCEL. It should then be sorted such that higher slope discrepancies are listed first. You 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 if the plot is jagged rather than straight or smoothly curved, the integrity of finite-difference derivatives for this observation is questionable. PEST's performance may therefore be degraded.

6.4 MULJCOSEN

MULJCOSEN reads a sequence of JCO files named *case.jco.N* where *N* represents a sequence of integers starting at 1. Such a sequence of JCO files is written by PEST if the JCOSAVEITN variable in the "control data" section of the PEST control file is supplied as "jcosaveitn". Gross variations of sensitivities between iterations may provide an indication of questionable model numerical performance.

MULJCOSEN is run using the command

```
muljcosen case obspar aname outfile
```

where

case	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 JCO files, and
outfile	is a file to which MULJCOSEN records the outcomes of its calculations.

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. Formulas through which composite observation and parameter sensitivities are computed are provided in section 5.3 of Doherty (2015). Both of these formulas involve observation weights. Hence as well as reading the sequence of Jacobian matrix files corresponding to the PEST case supplied on its command line, MULJCOSEN also reads the corresponding PEST control file, as well as any observation covariance matrix files that are cited in this file.

Composite sensitivities are written in tabular form to its output file. It is important to note that if PEST was run in "regularisation" mode, then regularisation observations/prior information equations are ignored in computation of composite parameter sensitivities. If PEST was run in "predictive analysis" mode, the prediction is likewise ignored.

7. Model Pre- and Postprocessing

7.1 Introduction

This chapter describes two utilities that perform similar, though different, roles. The first, PAR2PAR, is a "PEST-aware" model preprocessor while the second, OBS2OBS, is a "PEST-aware" model postprocessor. In normal usage, each would be run through a batch file which PEST runs as "the model". PAR2PAR undertakes parameter transformations while OBS2OBS undertakes model output transformations. In both cases the transformations which they undertake can be expressed using equations of arbitrary complexity.

7.2 PAR2PAR

7.2.1 General

On many occasions of model calibration there is a need to manipulate parameters before providing them to a model. There can be a number of reasons for this; two of them are now outlined.

Parameter Ordering

Suppose that a particular surface water or land use model has three parameters named *infilt1*, *infilt2* and *infilt3*. For purposes of illustration, let it be assumed that these parameters govern infiltration of water into different parts of a catchment, in this case into subareas 1, 2 and 3 respectively. Soil property data may suggest that infiltration increases with subarea index, that is that *infilt1* < *infilt2* < *infilt3*. Thus, during the parameter estimation process, it would be desirable for the lower bound of *infilt2* to be the current value for *infilt1*, and for the lower bound of *infilt3* to be the current value of *infilt2*.

Unfortunately it would be very difficult to incorporate parameter-dependent bounds into the PEST inversion algorithm. However an alternative path can be taken which accomplishes the same thing. This alternative path consists of estimating *infilt1* together with two other parameters named *infiltrat2* and *infiltrat3* ("*infiltrat*" stands for "infiltration ratio"). These latter two parameters are defined by the relationships

$$infiltrat2 = infilt2/infilt1 \qquad (7.2.1a)$$

and

$$infiltrat3 = infilt3/infilt2 \qquad (7.2.1b)$$

Desired infiltration parameter ordering relationships will be maintained if each of *infiltrat2* and *infiltrat3* is provided with a lower bound of 1.0 in the parameter estimation process implemented by PEST.

In using this device to ensure that correct infiltration parameter ordering relationships are maintained, PEST must work with parameters *infilt1*, *infiltrat2* and *infiltrat3*, while the model must be provided with parameters *infilt1*, *infilt2* and *infilt3*. The necessary parameter transformation process can be accomplished by running the utility program PAR2PAR as a model preprocessor contained in a "composite model" run by PEST as a batch file. PAR2PAR receives the current PEST-calculated values of *infilt1*, *infiltrat2* and *infiltrat3*; it then transforms these into values for *infilt1*, *infilt2* and *infilt3*. Then it writes one or more model input files (based on appropriate template files) containing the current values of these

native model parameters. Based on equations 7.2.1a and 7.2.1b, PAR2PAR must be programmed to calculate *infilt2* and *infilt3* using the relationships

$$infilt2 = infilt1 * infiltrat2 \quad (7.2.2a)$$

$$infilt3 = infilt2 * infiltrat3 \quad (7.2.2b)$$

Seasonal Parameter Variations

Some model parameters show seasonal variation. For environmental models which simulate water or crop-growth processes in agricultural areas, "crop factor" may be one such parameter. Crop factor is also a parameter that (together with other parameters) often requires adjustment through the calibration process in order that a land use model can replicate measured crop water usage, observed crop growth, or some other system response for which historical records are available.

Many models require that the crop factor be provided on a monthly basis. However while monthly crop factors may indeed require estimation through the inversion process, it would generally be unwise to attempt to estimate each monthly crop factor independently of every other monthly crop factor, for this would ignore an inherent relationship between these parameters, this being the fact that variation of crop factor with season may show a regular (perhaps sinusoidal) pattern. To ignore this pattern in parameterising the model would be to ignore an important facet of system behaviour. Furthermore, in many model calibration contexts, it would be unlikely that 12 different monthly crop factors could be independently estimated with any degree of uniqueness because of the high degree of correlation that is likely to exist between these individual parameters (especially where the data available for model calibration is limited).

For a case such as this, a suitable parameterisation strategy may be to estimate the mean monthly crop factor, together with the amplitude and phase of the seasonal variation of the crop factor about this mean. Thus twelve parameters are replaced by three. In implementing this strategy, PEST thus estimates three parameters while the model is provided with the twelve parameters which it requires. The task of transforming the three parameters estimated by PEST to the twelve parameters employed by the model can be accomplished using PAR2PAR as a model preprocessor, run by PEST just before the model on every occasion that the model is run. Once again, this can be accomplished by including both of the PAR2PAR and model executables in a batch file run by PEST as a "composite model". On the basis of the three parameters adjusted by PEST (named, for example, *mean*, *amplitude* and *phase*), PAR2PAR will calculate the monthly crop factor parameters required by the model (named, for example, *crop1, crop2…crop12*) using a series of relationships such as

$$crop1 = mean + amplitude * sin((1 + phase)*2.0*3.142/12.0) \quad (7.2.3a)$$

$$crop2 = mean + amplitude * sin((2 + phase)*2.0*3.142/12.0) \quad (7.2.3b)$$

etc.

In these equations *phase* is measured in months; as is explained below, the argument of the *sin* function must be supplied in radians, where 2π radians is equal to a full cycle.

Seasonal parameter variation can be expressed in a number of different ways; use of the *sin* function is just one of them. Another method would be to use "seasonal ratios"; if this is done, then only one parameter may require estimation, this being the factor by which all such ratios are multiplied to achieve a good fit with the calibration dataset.

7.2.2 Using PAR2PAR

Running PAR2PAR

PAR2PAR is run using the command

```
par2par infile
```

where

infile is a PAR2PAR input file which must be prepared by the user.

The PAR2PAR Input File

The structure of the PAR2PAR input file is shown in figure 7.1. An example of such a file is provided in figure 7.2.

```
* parameter data
PARNME = expression
PARNME = expression
.
.
* template and model input files
TEMPFLE INFLE
TEMPFLE INFLE
.
.
* control data
PRECIS DPOINT
```

Figure 7.1 Structure of the PAR2PAR input file.

```
* parameter data
infilt1 = 0.3456
infiltrat2 = 1.0453
infiltrat3 = 1.5432
infilt2= infilt1 * infiltrat2
infilt3 = infilt2 * infiltrat3
* template and model input files
model1.tpl model1.in
model2.tpl model2.in
* control data
single point
```

Figure 7.2 An example of a PAR2PAR input file.

A PAR2PAR input file must contain at least a “parameter data” section and a “template and model input files” section. The “control data” section is optional; if it is omitted, the default values of “single” and “point” are supplied for the variables PRECIS and DPOINT.

The “parameter data” section of the PAR2PAR input file provides the means whereby values are assigned to a set of parameters. These values can be provided either by the direct assignment of numbers, or through mathematical expressions. These expressions (which may be of considerable complexity) may cite parameters whose values were assigned in previous expressions.

The “template and model input files” section of the PAR2PAR input file provides the names of template files together with the names of the model input files to which they correspond. Once it has determined values for all parameters appearing on the left sides of the expressions listed in the “parameter data” section of its input file, PAR2PAR writes these parameter

values to the nominated model input files using template files based on these model input files (just like PEST does).

The following should be noted.

- Any parameter appearing in any of the template files listed in the "template and model input files" section of the PAR2PAR input file must be assigned a value in the "parameter data" section of the PAR2PAR input file.
- If there is more than one template/model input file pair listed in the "template and model input files" section of the PAR2PAR input file, any particular template file can be cited more than once if desired. However each model input file can be cited only once, for it would make no sense for a model input file generated on the basis of one template file to be overwritten by another model input file generated on the basis of the same or another template file.

If either of these rules is violated, PAR2PAR will inform you of this through an appropriate error message.

All template files cited in the "template and model input files" section of the PAR2PAR input file should be checked for correctness using TEMPCHEK. While PAR2PAR will detect and report any errors that it finds in these files, it will only report the first error that it encounters; then it will cease execution. TEMPCHEK, on the other hand, attempts to examine the entirety of a template file, reporting all errors to the screen. TEMPCHEK is documented elsewhere in this manual.

Parameter Relationships

The relationships through which parameter values are calculated from numbers, or from values previously assigned to other parameters, may be mathematical expressions of complex form. They can include any or all of the "*", "/", "+", "-" and "^" operators as well as brackets. (Note that the "^" operator raises the number in front of the "^" symbol to a power equal to the number trailing the "^" symbol; this operation can also be designated using the "**" symbol as in the FORTRAN programming language.) Mathematical operations of equal rank are evaluated in the order "^" followed by "*" and "/", followed by "+" and "-", as is the usual convention. This order can be overridden by the use of brackets.

The following mathematical functions are supported in expressions through which parameter values are calculated – *sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sinh*, *cosh*, *tanh*, *exp*, *log*, *log10*, *abs* and *sqrt*. Note the following rules governing use of these functions.

- As is the FORTRAN convention, the arguments of the trigonometric functions *sin*, *cos* and *tan*, and the values returned by their inverse functions *asin*, *acos* and *atan*, are assumed to be in radians. There are 2π radians in a circle; thus 2π radians are equal to 360 degrees.
- The *log* function is to base *e*; for logarithms to base 10, use the *log10* function.
- For some of the functions listed above, arguments must lie within a specific numerical range (for example the argument of the *log* function must always be greater than zero). If a function argument is provided which is outside of its legal range, PAR2PAR will often trap the error and cease execution with an appropriate error message. However in some rare instances the argument may "slip through" and a compiler-generated error message will be supplied upon termination of PAR2PAR execution.

The following rules apply when formulating mathematical expressions to calculate parameter values.

- Expressions may contain both numbers and parameters. However where a parameter is used, its value must have been calculated (or supplied) in a previous expression.
- As is the normal PEST convention, parameter names must be 12 characters or less in length.
- Spaces can be placed next to operators, brackets and functions. However they cannot appear within numbers, parameter names or function names.

Some examples of allowable mathematical expressions are provided in figure 7.3.

```
trans5 = k5 * (top5 - bottom5)
pi = 3.14159
par3 = 3.4 * (4.5 + trans5 ^ (3 + sin(0.6)))
par4 = par3 / (pi + exp(5.0 + par3/trans5))
par5 = -(par1 + par2) * cosh(pi * trans5)
```

Figure 7.3 Examples of mathematical expressions supported by PAR2PAR.

If an expression is long, it may be continued onto the next line by placing the "&" character at the beginning of that line. Thus the expression

*par5 = -(par1 + par2) * cosh(pi * trans5)*

is equivalent to

par5 =
& -(par1+par2)
*& * cosh(*
*& pi * trans5)*

Generation of Model Input Files

Once it has calculated values for all parameters, PAR2PAR writes these values to one or more model input files using templates of these files to govern parameter value placement. Use of template files for writing model input files is fully discussed in section 2.2 of part I of this manual. As is described in that section, slight variations of the way in which numbers representing parameter values are written to model input files can be effected through use of the PRECIS and DPOINT variables; values for these variables are supplied in the optional "control data" section of a PAR2PAR input file. If PRECIS is set to "single", numbers are written to model input files using the "E" character for exponentiation. However if it is set to "double", the "D" character is used; furthermore, if there is sufficient space, up to 23 characters can be used to record the value of the parameter instead of the usual maximum of 13. Setting the DPOINT variable to "nopoint" instructs PEST to write a parameter's value to a model input file without the decimal point if this can be accomplished through numerical formatting, thereby gaining one extra significant figure of precision. (More will be said about precision shortly.) As is stated above, the "control data" section of the PAR2PAR input file is optional; if it is omitted, default values of "single" and "point" are supplied for PRECIS and DPOINT respectively.

7.2.3 Using PAR2PAR with PEST

The Composite Model

As was discussed above, when used with PEST, PAR2PAR will normally be run as part of a

"composite model" encapsulated in a batch file. Thus whenever PEST runs the model, it first runs PAR2PAR (and any other model preprocessors cited in the batch file), followed by the model (followed by any model postprocessors cited in the batch file).

As for any other model executable program which uses parameters which require estimation by PEST, a template file must be built, based on a PAR2PAR input file. Just before it runs the model, PEST will then write current parameter values to the PAR2PAR input file using the corresponding template file. An example of such a template file, based on the PAR2PAR input file shown in figure 7.2, is provided in figure 7.4.

```
ptf $
* parameter data
infilt1 = $infilt1  $
infiltrat2 = $infiltrat2$
infiltrat3 = $infiltrat3$
infilt2= infilt1 * infiltrat2
infilt3 = infilt2 * infiltrat3
* template and model input files
model.tpl model.in
* control data
single point
```

Figure 7.4 A template for the PAR2PAR input file of figure 7.2.

Based on the template file of figure 7.4, before PEST runs the model it will replace the strings "$infilt1$", "$infiltrat2$", and "$infiltrat3$" with the current values of the respective parameters. Note that these parameters do not need to be named the same as the PAR2PAR parameters to which values are assigned in the pertinent expressions in the PAR2PAR input file. They could have been given any name at all; the same parameter names are used by both PEST and PAR2PAR in this example simply as a matter of convenience. Furthermore, parameter spaces in the template of a PAR2PAR input file do not need to be restricted in their location to the right side of expressions comprised of a "=" symbol followed by a single number. See, for example, the PAR2PAR input file and corresponding template file depicted in figures 7.5 and 7.6. These accomplish the same task as the files depicted in figures 7.2 and 7.4.

```
* parameter data
infilt1 = 0.3456
infilt2= infilt1 * 1.23983
infilt3 = infilt2 * 1.53953
* template and model input files
model.tpl model.in
```

Figure 7.5 A PAR2PAR input file.

```
ptf $
* parameter data
infilt1 = $infilt1  $
infilt2= infilt1 * $infiltrat2$
infilt3 = infilt2 * $infiltrat3$
* template and model input files
model.tpl model.in
```

Figure 7.6 A template for the PAR2PAR input file of figure 7.5.

It is apparent that when using PAR2PAR as part of a composite model run by PEST there are two sets of template files involved in the inversion process, namely that used by PEST to write a PAR2PAR input file, and those used by PAR2PAR to write model input files. These

should not be confused. PEST should never be instructed to use a template file to write a model input file that is also cited in the "template and model input files" section of a PAR2PAR input file. If this happens, the model input file generated by PEST will be overwritten by that generated by PAR2PAR.

It often happens that only a few parameters required by a model need to be calculated by an expression cited in a PAR2PAR input file; other model parameters can be estimated directly by PEST. These latter parameters can simply be "passed through" PAR2PAR by assigning them numerical values in the pertinent expressions in the PAR2PAR input file. Figure 7.7 shows a PAR2PAR input file in which only parameter *par8* is calculated through manipulation of other parameters; figure 7.8 shows the corresponding template file of the PAR2PAR input file. Parameters *par1* to *par5* are passed directly to the model through the template file *model.tpl* of the model input file *model.in*. The template file *model.tpl* thus cites all of parameters *par1* to *par5* as well as parameter *par8*. (It may also cite *par6* and *par7*.)

```
* parameter data
par1 = 1.583745e-4
par2 = 5.395832e-1
par3 = 4.583924e-2
par4 = 5.389028e-5
par5 = 4.389428e-2
par6 = 3.559313e-1
par7 = 5.395355e-2
par8 = par6 * exp(-par7)
* template and model input files
model.tpl model.in
```

Figure 7.7 A PAR2PAR input file.

```
ptf $
* parameter data
par1 = $par1      $
par2 = $par2      $
par3 = $par3      $
par4 = $par4      $
par5 = $par5      $
par6 = $par6      $
par7 = $par7      $
par8 = par6 * exp(-par7)
* template and model input files
model.tpl model.in
```

Figure 7.8 A template file for the PAR2PAR input file of figure 7.7.

Numerical Precision

As is explained in part I of this manual, when PEST writes a number to a model input file on the basis of a template file, it alters its internal representation of that number to account for the fact that the number may be written to the model input file with less than the maximum number of significant figures with which that number can be represented internally within the computer. Thus when PEST calculates derivatives of model outputs with respect to parameters using finite differences, the differences between incrementally-varied parameter values will be exactly correct because both PEST and the model use exactly the same parameter values.

The ability for PEST to compensate for limited parameter space widths on model input files is lost when parameter values are written to those files using program PAR2PAR (because

PEST has no way of adjusting its internal representation of parameters based on PAR2PAR outputs). Thus unless the formatting requirements of the model input file are such that it allows model input parameters to be supplied with full numerical precision (which is normally about 7 significant figures), slight errors will be incurred in the derivatives calculation process. (Note that where a number is small or large enough for exponential notation to be required for its representation, up to 14 characters may be required for the representation of that number using 7 significant figures.) Imprecision in derivatives calculation can have a profound effect on the outcome of an inversion process. *Thus you should make absolutely sure that the template files used by PAR2PAR to write model input files use parameter space widths which are as large as the model will tolerate (up to a maximum of 14 characters if using single precision arithmetic, or 23 characters if using double precision arithmetic).* If model input file formatting requirements are too restrictive to allow a parameter value to be written without some loss of significance, then you should at least be aware of the fact that use of PAR2PAR under these circumstances has the potential to reduce the efficacy of PEST's performance.

Intermediate Files

Before it runs the model, PEST deletes all model output files that it knows about (i.e. the model output files cited in the PEST control file). Hence if the model fails to run, PEST will not read old model output files produced on previous model runs, mistaking them for new ones. Thus if PEST generates an error message saying that it cannot find a particular model output file, this is a sure sign that, for some reason, the model failed to run. In most cases the matter is then easily rectified by taking some simple measure such as altering the contents of the "model command line" section of the PEST control file.

Where a model is comprised of multiple executable programs listed in a batch file, similar considerations apply to "intermediate model files", i.e. to files generated by one or more of the executable programs comprising the composite model and read by one or more succeeding executable programs cited in the model batch file. If, for some reason, an executable program which generates such an intermediate file fails to run, then later executable programs of the composite model may read old intermediate files, mistaking them for new ones. If this happens, model outputs will not reflect current parameter values; in fact, because they are independent of current parameter values, PEST will probably declare that at least some model outputs are insensitive with respect to some parameter values. This problem can be avoided if commands are included in the model batch file to delete all intermediate files before any of the executable programs comprising the model are run. If PAR2PAR is one such executable program, then all model input files cited in the "template and model input files" section of the PAR2PAR input file should be deleted prior to running PAR2PAR. Figure 7.9 shows an example of a model batch file in which this precaution is taken.

```
rem Model input files written by PAR2PAR are deleted.
del model1.in
del model2.in
rem PAR2PAR is run.
par2par par2par.in
rem The model is run.
model
```

Figure 7.9 A model batch file which includes PAR2PAR as one of the model executable programs.

In the batch file depicted in figure 7.9, file *par2par.in* is the PAR2PAR input file. If it is

desired that screen output from all programs comprising the composite model (including the model batch file itself) be suppressed so that the model's screen output does not interfere with that of PEST, the batch file shown in figure 7.9 could be altered to that shown in figure 7.10. (Note that on a UNIX system "> nul" should be replaced by "> /dev/null".)

```
@echo off
rem Model input files written by PAR2PAR are deleted.
del model1.in > nul
del model2.in > nul
rem PAR2PAR is run.
par2par par2par.in > nul
rem The model is run.
model > nul
```

Figure 7.10 The batch file of figure 7.9 with all screen output suppressed.

7.3 OBS2OBS

7.3.1 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.

7.3.2 OBS2OBS Input File

Before running OBS2OBS an input file must be prepared. An example of an OBS2OBS input file is shown in figure 7.11.

```
* 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
```

Figure 7.11 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 in figure 7.11, 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" Section

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.

- An instruction filename cannot be repeated;
- Observation names cited in instruction files must be unique;
- Observation names must not contain spaces; their length is limited to 20 characters.

"Equations" Section

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 table 7.1.

Operators	^, /, *, -,+, (,)
Functions	abs, acos, asin, atan, cos, cosh, exp, log, log10, sin, sinh, sqrt, tan, tanh, neg, pos, min, max, mod

Table 7.1 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" Section

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 are 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 in figure 7.12.

```
head1                    3.456755261693154
head2                    4.324500931552616
hd1                      0.000000000000000
conc1                    30.45670000000000
conc2                    53.23432315324721
cd1                      22.77762050136947
```

Figure 7.12 Example of an OBS2OBS output file.

The following features of an OBS2OBS output file are salient.

- The OBS2OBS output file contains two columns. No column headers are provided.
- 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.
- 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.

7.3.3 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 observed counterparts to OBS2OBS-generated quantities to a PEST calibration dataset. In this instruction file observations are given the same names as variables read by OBS2OBS from model output files or named in equations 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.

7.3.4 Some Uses of OBS2OBS

Observation Postprocessing

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. As described by Doherty (2015), this strategy can protect parameter estimates from calibration-induced bias if properly undertaken. 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 for the upper bound 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 "hard" bounds to be enforced 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 parent 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. An alternative approach may be to use parameter thresholds in the same manner as that shown above for observations instead of introducing a penalty as soon as the parameter deviates from some central value. The upper and lower parameter thresholds (both of which should be within the bounds associated with that parameter) can be used to define 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 approached, then bounds enforcement takes place in the usual way.

To implement soft bounds, 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.

7.3.5 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 own 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 was discussed in documentation of PAR2PAR, it is a good idea to delete the input file of any executable program that runs in a 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, on the occasion of any model run, this input file does not get written 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 as it is when using PAR2PAR (or any other model batch file comprised of multiple executable programs), 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 you feel uncomfortable in straying from the established principle of deleting all input files used by model postprocessors (this being a worthy principle), then you 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 model batch file.

8. PEST Statistical Postprocessing

8.1 Introduction

This chapter describes a number of utilities that can be used after a PEST run in order to gain a better understanding of the information content of the calibration dataset, and of the estimability (or otherwise) of individual parameters. Some of these utilities (EIGPROC, PCOV2MAT, INFSTAT and INFSTAT1) are best suited for use in the well-posed inversion context, where all parameters are estimable on the basis of information contained in the calibration dataset. Others utility programs documented in this chapter (SUPCALC, IDENTPAR and SSSTAT) embrace the ill-posedness of an inverse problem; they calculate the dimensionality of the solution space, and the degree to which each parameter lies within or without of this space.

Further insights can be gained into the well-posedness or otherwise of an inverse problem, and into the information content of the calibration dataset as it pertains to each parameter, through use of the GENLINPRED utility. However GENLINPRED is not documented in this section as its capabilities extend to analysis of the uncertainties of predictions, and not just of parameters. Predictive uncertainty and error variance analysis is the subject of another chapter of this manual.

8.2 EIGPROC

8.2.1 General

EIGPROC reads a PEST run record file and a PEST sensitivity file. It extracts and summarises information from these files. Recall that a PEST run record file is always named *case.rec* where *case* is the filename base of the PEST control file; a sensitivity file is named *case.sen*.

Use of EIGPROC (which stands for "eigenstuff processor") is predicated on the following assumptions.

- PEST has been run in "estimation" mode.
- The ICOV, ICOR and IEIG variables in the "control data" section of the PEST control file are all set to 1.
- PEST has run to completion, or has been stopped using the "stop with statistics option". (This will happen if the PSTOPST program has been run in another window).
- The parameter estimation problem is well posed so that the $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix (often referred to as the "normal matrix") inverted by PEST to obtain the parameter covariance matrix is not singular; thus a post-calibration covariance matrix (and associated correlation coefficient and eigendata matrices) are recorded at the end of the run record file.

8.2.2 Using EIGPROC

EIGPROC is run using the command

```
eigproc case N exlim outfile
```

where

case	is the filename base of the PEST control file,
N	is the number of eigenvalues to be processed,
exlim	is the eigencomponent exclusion limit, and
outfile	is the EIGPROC output file.

EIGPROC's input requirements will now be explained in more detail.

Where PEST is run in "estimation" mode, and where an inverse problem is well-posed, the post-calibration covariance matrix is computed using equations 5.2.13 and 5.2.18 of Doherty (2015). The largest eigenvalues of the post-calibration parameter covariance matrix are associated with eigenvectors that describe parameter combinations that contain most of the uncertainty of an inferred parameter set. By supplying an appropriate value for *N* (which should be equal to, or less than, the number of adjustable parameters), you inform EIGPROC how many eigenvalues you would like listed in the EIGPROC output file. Eigenvalues are counted starting from the highest.

For a given eigenvalue, only those parameters whose components of the corresponding eigenvector which are significantly nonzero are of special interest, for these are the parameters which, by virtue of insensitivity and/or correlation, are estimated with larger uncertainty than other parameters. If the eigencomponent exclusion limit supplied on the EIGPROC command line is, for example, 0.1 then only parameters whose absolute eigenvector components are greater than 0.1 are listed with the information supplied for a particular eigenvalue in the EIGPROC output file. (Eigenvectors of the post-calibration covariance matrix are normalized so that their magnitudes are all 1.0). The name of the EIGPROC output file is supplied as the last of EIGPROC's command line arguments.

Once it has parsed its command line, EIGPROC reads the PEST run record file, followed by the PEST sensitivity file. For each eigenvalue (starting from the highest) it provides information such as that shown in figure 8.1.

```
Eigenvalue number: 14     Value =  0.24990     -------->

 Parameter    Eigenvector      Sensitivity
              component
  v3pp42         0.496          9.071E-03
  v2pp42         0.487          1.060E-02

 Correlation coefficient matrix for these parameters:-
                v3pp42       v2pp42
 v3pp42           1.0          0.84
 v2pp42           0.84          1.0
```

Figure 8.1 Part of an EIGPROC output file.

For each eigenvalue, EIGPROC lists parameters in decreasing order of eigenvector component magnitude; the actual eigenvector component is also listed, together with the composite parameter sensitivity as read from the sensitivity file. (Note that these sensitivity values are extracted from the end of the sensitivity file where statistics related to optimised parameters are listed.) Parameters are only listed in this table if their eigencomponent magnitude is greater than the "eigencomponent exclusion limit" supplied on the EIGPROC command line.

Underneath the eigendata a correlation coefficient matrix is recorded. This is a sub-matrix of the parameter correlation coefficient matrix listed in the PEST run record file, featuring only those parameters which appear in the above eigendata listing.

8.3 PCOV2MAT

PCOV2MAT extracts a PEST-calculated post-calibration covariance matrix from files recorded by PEST. It re-writes this matrix in PEST matrix file format. Once in that format, it can be processed using matrix utilities documented elsewhere in this manual.

Suppose that the filename base of the PEST control file is *case*. PEST records a post-calibration covariance matrix file at the end of its run record file (named *case.rec*) and also in a file named *case.mtt*; the latter file is refreshed during every iteration of the inversion process. However PEST only records a post-calibration covariance matrix in either of these files if the following conditions are met.

- PEST is run in "estimation" mode;
- The ICOV variable in the "control data" section of the PEST control file is set to 1;
- Neither singular value decomposition nor LSQR is employed in solution of the inverse problem;
- The $\mathbf{J}^t\mathbf{QJ}$ matrix (where **J** is the Jacobian matrix and **Q** is the weight matrix) is invertible.

See section 5.2.2 of Doherty (2015) for a discussion of the post-calibration covariance matrix produced through solution of a well-posed inverse problem.

PCOV2MAT is run using the command

```
pcov2mat 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 re-written; see section 2.4 for format specifications for this type of file.

The following should be noted.

1. Only adjustable parameters are cited in a post-calibration parameter covariance matrix; fixed and 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 optimised parameters, whereas the covariance matrix recorded in the latter file is calculated on the basis of sensitivities pertaining to the latest PEST iteration.

Once the post-calibration parameter 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 5.2.17 of Doherty (2015) using the MATQUAD utility documented elsewhere in this manual. Prediction sensitivities required for this calculation 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.

8.4 INFSTAT

8.4.1 General

The INFSTAT utility computes a number of observation influence statistics, namely

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. See also section 5.3.3 of Doherty (2015) where the influence statistics computed by INFSTAT are defined, and where formulas for their computation are provided.

INFSTAT should be used for computation of influence statistics where an inverse problem is well-posed. While it will tolerate usage of PEST in "regularisation" mode this is not recommended, as inverse problem ill-posedness compromises the theory on which computation of these influence statistics is based. Where an inverse problem is ill-posed, consider using INFSTAT1 in place of INFSTAT. Better still, consider using utility programs such as PREDUNC5 and PREDVAR5 (documented elsewhere in this manual) which accommodate the existence of a null space in assessing data worth. See also the SSSTAT utility.

8.4.2 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.

INFSTAT reads the PEST control file *case.pst*, the Jacobian matrix file *case.jco*, and the residuals file *case.res* for a completed PEST run. Where the completed PEST run was in "regularisation" mode, INFSTAT also reads the PEST-produced resolution data file *case.rdf* in order to obtain optimised regularisation weights.

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 in influence statistics calculations can be limited to *nsig* by the user. If omitted (which will normally be the case), then elements of the Jacobian matrix are represented with the precision with which they are read from the JCO file when INFSTAT calculates its influence statistics.

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" or "pareto" modes.
2. Observations and prior information equations with weights of zero are ignored.
3. As noted in part I of this manual, it is possible that estimates of parameters may have improved slightly on the final iteration of the inversion process undertaken by PEST after calculation of the set of parameter sensitivities which occupy the Jacobian matrix file (i.e. the JCO file). If the resulting mismatch between parameter values and parameter sensitivities is seen as a problem, the PARREP utility can be employed to build a new PEST control file with optimised parameter values supplied as initial parameter values in this file. If NOPTMAX is set to -1 in this file, PEST will calculate a new Jacobian matrix using these parameters, record this matrix in a new JCO file, record some statistical information on its run record file, and then cease execution. (If

you wish, you can even ask PEST to calculate the Jacobian matrix using a higher order finite-difference stencil on this dedicated PEST run.) 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 an observation 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

$$\mathbf{j} = \mathbf{C}(\boldsymbol{\varepsilon})^{-1/2}\mathbf{h} \qquad (8.4.1)$$

where $\mathbf{C}(\boldsymbol{\varepsilon})$ 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 measurement 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 do have when PEST is run in "regularisation" mode. Nevertheless, they can still be calculated; their meaning is left to the user to interpret.

6. Where singular value decomposition was employed for solution of the inverse problem, INFSTAT does not compute influence statistics. This is because it is not aware of the truncation point used for separation of solution and null spaces. If the PEST control file has a "singular value decomposition" section, and if SVDMODE is non-zero in that section, INFSTAT ceases execution with an appropriate error message.

7. Where SVD-assist was implemented for solution of the inverse problem, INFSTAT does indeed compute influence statistics (provided singular value decomposition was not used to calculate values for super parameters). However you should bear in mind that these statistics 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 using the magnitude of these statistics with respect to thresholds as outlined in the references cited above. It should be noted that the establishment of a threshold is somewhat arbitrary (Hadis 1992); in the case of the Hadis statistic, the relative influence of observations can have a large effect on calculation of this threshold because the latter relies on the mean and standard deviation of influence values. If one or more observations are orders of magnitude more influential than the others, use of a thus-calculated threshold may remove observations that are nevertheless 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 the equation used for its calculation), a value of -1.0E35 is recorded at 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 accurate.

8.5 INFSTAT1

8.5.1 General

INFSTAT1 performs a similar function to that of INFSTAT. However it is better equipped to accommodate inverse problem ill-posedness than INFSTAT. In spite of this, its use does not overcome a fundamental problem with the type of influence statistics that INFSTAT and INFSTAT1 support. This is the fact that these statistics do not quantify the ability of data to reduce the dimensionality of the calibration null space. In many calibration contexts this is the best measure of the worth of data. As stated in documentation for INFSTAT in the previous subsection, utility programs such as PREDVAR5 and PREDUNC5 can overcome this problem. See also the SSSTAT utility which can likewise accommodate problem ill-posedness while calculating indicators of data worth.

Where an inverse problem is ill-posed, INFSTAT cannot invert the sensitivity-based matrices required for calculation of its information statistics. This situation can be remedied by introducing Tikhonov regularisation to the inverse problem, for example through the use of prior information expressing expert knowledge. As is discussed in part I of this manual, PEST calculates its own weights for observations and prior information equations comprising regularisation constraints. The INFSTAT utility described in the previous section tries to use these weights. INFSTAT1 does not.

Instead of relying on Tikhonov regularisation that may have been used in a PEST inversion problem to achieve the matrix invertibility that calculation of its statistics requires, INFSTAT1 employs subspace methods as encapsulated in singular value decomposition to achieve this same end. 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 singular value truncation should take place, and hence of the dimensionality of the calibration 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 described below can be used to obtain estimates of optimal solution space dimensionality. Use the lower of the two values suggested by SUPCALC.

As described by Doherty (2015), and as stated above, when singular value decomposition is employed as a regularisation device, the inverse problem is re-formulated to estimate the projection of a real-world parameter set onto the (often small-dimensional) solution space of

the calibration problem. In PEST parlance, the square of the direction cosine between an individual parameter and its projection into the calibration solution space is referred to as the "identifiability" of that parameter; this can be calculated using the IDENTPAR, SSSTAT and GENLINPRED utilities. In a typical calibration context the identifiability of many parameters may not be very high. Where this is the case, the estimated values of these parameters may have large errors due to their high projection onto the calibration 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 calibration solution space. This, indeed, is the weakness of the statistics supported by INFSTAT and INFSTAT1 in assessing data worth.

8.5.2 Using INFSTAT1

INFSTAT1 is run using the command

```
infstat1 case outfile nsing
```

where

case	is the filename base of a PEST control file,
outfile	is the INFSTAT1 output file, and
nsing	is the singular value truncation threshold.

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.

As for INFSTAT, PEST must have been run prior to using INFSTAT1 as a Jacobian matrix corresponding to the nominated PEST control file must exist (in a file named *case.jco*). If PEST is run specifically to obtain this matrix, then NOPTMAX must be set to -1 and not -2 in the PEST control file so that PEST records a residuals file on termination of execution. INFSTAT1 uses these residuals in calculation of its influence statistics.

8.6 SUPCALC

8.6.1 General

"SUPCALC" stands for "super parameter calculation". It was originally written to assist in choosing the minimum number of super parameters to employ in SVD-assisted inversion. (Recall, however, from part I of this manual that the *optimum* – as distinct from *minimum* - number of super parameters to employ is "as many as possible".) However its use extends beyond this. It is able to provide an indication of the optimal dimensionality of the solution space for a particular inverse problem. This can be loosely equated to the number of pieces of accessible information in a calibration dataset. Each of these pieces of information provides the basis for estimation of the scalar projection of a real world parameter set onto an orthogonal axis in parameter space formed though singular value decomposition of the weighted Jacobian matrix $\mathbf{Q}^{1/2}\mathbf{J}$ associated with the present inverse problem.

The theory on which SUPCALC is based is described in section 6.2.5 of Doherty (2015). See in particular section 6.2.5.3. As described in that section, SUPCALC tries to determine the singular value at which estimation of the corresponding parameter solution space projection may introduce more potential for error than estimation of that projection on the basis of expert knowledge alone. As section 6.2.5 of Doherty (2015) shows, as singular values decrease in value, measurement noise is amplified in estimating corresponding parameter

solution space projections onto the columns of the $\mathbf{V}_1$ matrix discussed in that section, this contributing to a growing potential for estimation error. (This is the phenomenon of "over-fitting".) At some point it is amplified so much that there is smaller potential for error in estimating the parameter projection through expert knowledge alone than on the basis of the calibration dataset. This is the point that SUPCALC tries to determine.

Actually, SUPCALC provides suggestions for both the upper and lower bounds of the singular value truncation point. The lower bound is calculated in the manner described above. The upper bound is marked by the singular value at which the ratio of highest to lowest squared singular value of the weighted Jacobian matrix is equal to 10^{-7}. This is the point at which numerical noise associated with the Jacobian matrix is likely to be amplified to the point at which estimation of parameter projections is invalidated. (Selection of this truncation point assumes that finite parameter differencing incurs a relative error of between 10^{-3} and 10^{-4} for estimates of partial derivatives embodied in the Jacobian matrix.)

As is discussed by Doherty (2015), any estimate of post-calibration predictive or parameter error variance relies on two covariance matrices. These are C(**k**), the covariance matrix of prior expert knowledge of parameters, and C(**ε**), the covariance matrix of measurement noise. SUPCALC can read the former from a parameter uncertainty file. Alternatively, if you have not prepared such a file, it will calculate an approximation to this matrix by assuming the following.

- All parameters are statistically independent from a prior point of view.
- The standard deviation of each parameter is 0.3 times the difference between its upper and lower bound as recorded in the PEST control file; log-transformation status of pertinent parameters is taken into account in making this calculation.

(Note that if you use this option then you may need to give more thought than you otherwise would to the bounds that you supply for parameters in a PEST control file.)

SUPCALC attempts to make calculation of an approximate C(**ε**) matrix similarly easy. It asks for an estimate of the objective function (measurement objective function if Tikhonov regularisation is employed) that will be achieved through the inversion process. It divides this estimate by the number of non-zero-weighted, non-regularisation observations featured in the current inverse problem to obtain an estimate of the reference variance σ^2_r for the current problem. Following equation 5.2.6 of Doherty (2015), C(**ε**) is then calculated as

$$C(\boldsymbol{\varepsilon}) = \sigma^2_r \mathbf{Q}^{-1} \qquad (8.6.1)$$

Where **Q** is the weight matrix featured in the inverse problem.

8.6.2 Using SUPCALC

SUPCALC is run by typing its name at the screen prompt. Unlike many of the PEST utilities documented in this manual, 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. Instead, it communicates with the user through a series of prompts, to which appropriate responses must be supplied.

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 (i.e. JCO file) already exists. This control file may or may not include Tikhonov regularisation. If it does, all regularisation observations and prior information equations are

ignored by SUPCALC.

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 the dimensions of the calibration solution space are greater than is actually sustainable by the data. If your answer is too high then SUPCALC may underestimate the dimensions of the calibration solution space. However, given the approximate nature of SUPCALC's calculations, and the fact that in most modelling contexts the level of fit is determined more by so-called "structural noise" than by measurement noise, these considerations are probably of secondary importance. Furthermore, in many real-world modelling contexts the boundary between the solution and null spaces is "soft" because the minimum of the parameter projection error variance curve that SUPCALC seeks to find is difficult to establish; this is because the number of singular values over which near-minimal values are calculated is often quite broad.

SUPCALC next asks

```
To conduct SVD on Q^(1/2)X  - enter 1
To conduct SVD on XtQX      - enter 2
Enter your choice:
```

In these questions **X** is the Jacobian matrix **J**. The outcome of SUPCALC's deliberations should not depend on your choice; however SUPCALC execution time may depend on which of these options you select. If in doubt, choose the first option as it is often much quicker.

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 prior standard deviation of every parameter as the difference between its bounds (as read from the PEST control file) 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 the bounds interval signifies the 95 percent confidence interval, the standard deviation is the range divided by 4 (which is equivalent to multiplication by 0.25). In implementing this option, SUPCALC assumes that all parameters are statistically independent, and thus that the C(**k**) matrix is diagonal.

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:
```

Specifications for a parameter uncertainty file are provided in section 2.5 of this manual. Through this means a C(**k**) matrix of arbitrary complexity can be provided.

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 orthogonal unit vector represented in the **V** matrix obtained through singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ or $\mathbf{J}^t\mathbf{Q}\mathbf{J}$, starting with the highest singular value, SUPCALC computes the loss of error variance accrued through inclusion of this eigencomponent in the calibration solution space, and the gain in uncertainty incurred through the effect of

measurement noise in estimating it. It then sums the two of these. If the summation is negative (indicating a net loss of estimability of the parameter projection onto this direction in parameter space), SUPCALC recommends that this eigencomponent 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 estimated parameter error variance is encountered in this fashion as the singular value number is increased, the rise in total estimated parameter error variance will be even greater for eigencomponents corresponding to subsequent singular values. However where the C(**k**) matrix is non-diagonal, and where its diagonal elements are very different from each other, it is not impossible for falls in estimated parameter error variance to follow rises. Hence SUPCALC supplies its minimum recommended number of super parameters as the singular value number corresponding to the last eigencomponent for which inclusion in the solution space results in a net diminution of post-calibration parameter error variance.

As well as providing a value for the minimum number of dimensions of the solution space, SUPCALC also provides a recommendation for the maximum possible dimensions of the solution space. As stated above, the latter corresponds to the number of the last singular value of $\mathbf{J}^t\mathbf{QJ}$ that is above 10^{-7} of the first singular value of this matrix. 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 eigencomponents 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

Figure 8.2 A SUPCALC output file.

The first column of the SUPCALC output file lists singular values of $\mathbf{J}^t\mathbf{QJ}$; these are the squares of singular values of $\mathbf{Q}^{1/2}\mathbf{J}$. They are listed in order of decreasing singular value. The fall in error variance of estimation of each parameter projection from its pre-calibration level (the latter is equal to its pre-calibration error variance determined by C(**k**) alone) is provided in the second column. The rise in error variance accrued from the fact that data on which basis this parameter projection is estimated is contaminated by measurement noise is provided in the third column. The final column contains the fall in error variance minus the rise in error variance; if this is negative, there is a net rise in error variance, from which it must be concluded that the value of the parameter's projection onto the direction defined by the corresponding column of **V** is not worth including in the parameter estimation process. This defines the boundary of the calibration solution space.

8.7 IDENTPAR

8.7.1 General

Doherty (2015) and Doherty and Hunt (2009) define the identifiability of a parameter as the square of the cosine of the angle between a parameter and its projection onto the calibration

solution space. It is also the magnitude of the diagonal element of the resolution matrix corresponding to that parameter. See section 7.2.1 of Doherty (2015) for details.

If the identifiability of a parameter is 1.0, then that parameter is completely estimable on the basis of the current calibration dataset. This does not mean that its estimation is without error; however it means that measurement noise, and not an information deficit in the calibration dataset, is responsible for this error. Alternatively, if a parameter has an identifiability of 0.0, then the calibration dataset is completely uninformative of that parameter; thus the parameter is completely insensitive as far as the calibration dataset is concerned. On the other hand, if the identifiability of a parameter is between 0.0 and 1.0 then information within the calibration dataset that pertains to that parameter is shared between it and other parameters; the parameter can therefore not be resolved uniquely.

The value calculated for the identifiability of a parameter may depend on the number of dimensions attributed to the calibration solution space. The boundary between calibration solution and null spaces is often a soft boundary. Calculation of the location of this boundary may be assisted through the use of programs such as SUPCALC, SSSTAT and PREDVAR1 described elsewhere in this manual. The dependence of the identifiabilities computed for some parameters on the location of this often ill-defined boundary may erode the value of identifiability as a useful post-calibration statistic in some modelling contexts. In these contexts the relative parameter uncertainty variance reduction computed by GENLINPRED (which also varies between 0.0 and 1.0) may prove a superior statistic.

In spite of its drawbacks, the concept of parameter identifiability has an intuitive appeal. Furthermore, as the following discussion shows, some ancillary data on flow of information from the calibration dataset to the parameter solution space is also available as a by-product of its use.

8.7.2 Using IDENTPAR

IDENTPAR is run using the command

```
identpar case N vecfilebase matfile identfile [/s or /r]
```

where

case	is the filename base of a PEST control file,
N	is the number of dimensions comprising 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 of $\mathbf{J}^t\mathbf{QJ}$, (the default), while
/r (optional)	instructs IDENTPAR to undertake singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$.

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 in the "observation groups" section of the PEST control file if these are employed for any observation groups instead of weights). It then reads the Jacobian matrix corresponding to this PEST control file. As usual, this file is assumed to

have the same filename base as the PEST control file but to possess an extension of "*.jco*"; it is thus assumed that this file exists and has been computed either on the basis of initial parameter values recorded in the PEST control file (set NOPTMAX to -1 or -2 to do this) or during a prior parameter estimation process.

IDENTPAR next forms the $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrix (or $\mathbf{Q}^{1/2}\mathbf{J}$ 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 $\mathbf{V}_1$ (see Doherty, 2015). Suppose that the user-supplied basename for these files is *vfile*. Then the files will be named *vfile1.vec, vfile2.vec…..vfileN.vec*. Any of these files can be used as a predictive sensitivity file by any of the PREDVAR-suite or PREDUNC-suite utilities described elsewhere in this manual.

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. It also records the square of the magnitude of the projection of each unit parameter vector onto each eigencomponent comprising columns of the $\mathbf{V}_1$ matrix. These are the eigencomponent squared cosines defined by equation 7.2.5 of Doherty (2015). The pertinent element in the "identifiabililty" column in this same file is the square of the magnitude of the projection of each unit parameter vector onto the totality of this space as spanned by all solution space eigencomponent vectors. It is thus the sum of the preceding columns.

Interesting and informative plots can be produced by importing *identfile* into a spreadsheet or graphing program. For example figure 8.3 shows a bar chart of parameter identifiabilities, with the contribution from each solution space eigencomponent 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.

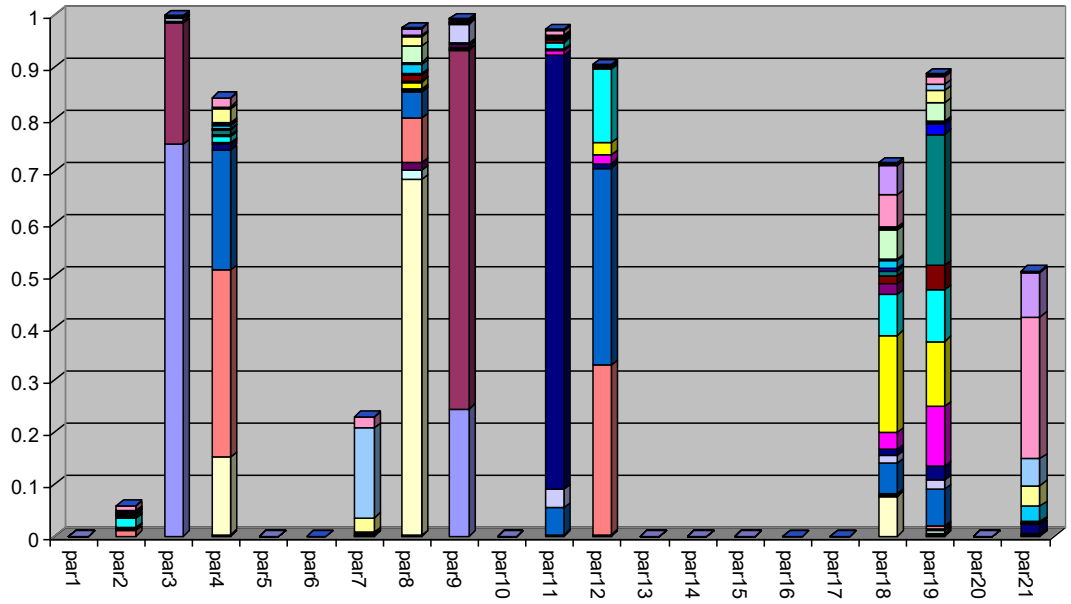


Figure 8.3 Bar chart of parameter identifiabilities coloured by contributions made to total identifiability by different solution space eigencomponents.

Plots such as that provided in figure 8.3 can be made even more informative if warmer colours (reds and yellows) are ascribed to projections onto eigencomponent vectors (i.e. unit vectors $\mathbf{v}_i$ comprising columns of the $\mathbf{V}_1$ matrix) associated with high singular values and cooler colours (blues and greens) are ascribed to projections onto eigencomponent vectors associated with low singular values. As discussed by Doherty (2015), measurement-noise-induced errors associated with estimated values of parameter eigencomponent projections increase as singular values associated with eigencomponents decrease. Warmer colours may therefore be indicative of smaller post-calibration error variance.

The following should be noted.

1. Weights used in computation of $\mathbf{Q}^{1/2}\mathbf{J}$ (or $\mathbf{J}^t\mathbf{Q}\mathbf{J}$) are those provided in the cited PEST control file.
2. Use of the $\mathbf{Q}^{1/2}\mathbf{J}$ and $\mathbf{J}^t\mathbf{Q}\mathbf{J}$ matrices should lead to identical identifiabilities. However if many parameters and/or observations are featured in the inversion process then the "/r" option should be used in running IDENTPAR as its execution will probably then be faster.
3. If PEST is run in "regularisation" mode, observations and prior information equations belonging to regularisation groups are ignored by IDENTPAR.

8.8 PARAMID

8.8.1 General

PARAMID is an old program whose use is no longer recommended. However it is still retained in the PEST suite as it may be useful to some.

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 inversion process to the eigenvectors spanning the calibration solution space. (The eigenvectors are the columns of the $\mathbf{V}$ matrix emerging from

singular value decomposition of the weighted Jacobian $\mathbf{Q}^{1/2}\mathbf{J}$ matrix.)

Use of PARAMID is based on the following premises.

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 "singular value decomposition" section of the PEST control file, thus ensuring that the full eigenvector matrix was recorded in the SVD file generated by PEST under these circumstances.
3. The initial Marquardt lambda was set to zero (and NUMLAM set to 1) in the "control data" section of the PEST control file.
4. At least one iteration was carried out before termination of PEST execution. Thus NOPTMAX was set to 1 or greater in the "control data" section of the PEST control file.

Note that if model run times are long and a Jacobian matrix file already exists based on a PEST run in which a strategy other than singular value decomposition was used to solve the inverse problem, then there is no need to undertake a full PEST run in order to obtain the SVD file. Simply alter the PEST control file so that the above conditions are met (after using the PARREP utility to place optimised parameter values into this file). Then start PEST using the "/i" switch. PEST will then prompt the user for the name of an existing Jacobian matrix file rather than calculating the Jacobian matrix itself.

8.8.2 Using PARAMID

PARAMID is run using the command

```
paramid case outfile
```

where

case is the filename base of an existing PEST control file which meets 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; it assumes that this is the dimensionality of the calibration solution space. For each adjustable parameter listed in the PEST control file, PARAMID then 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 the following.

1. All eigenvectors computed by singular value decomposition 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 estimability.

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 inversion process (and is probably insensitive).

2. If a parameter makes a moderate contribution to only one solution space eigenvector, then it may still not be completely 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.

8.9 SSSTAT

8.9.1 General

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.

For most of the utilities documented in this manual, reference is made to Doherty (2015) for the theory on which the respective algorithm is based. An exception is made for the SSSTAT utility so that all of this theory can be readily accessible in one place. That theory is now presented.

8.9.2 Theory and Concepts

Let the vector $\mathbf{h}$ represent measurements comprising the calibration dataset and let the vector $\boldsymbol{\varepsilon}$ denote errors associated with those measurements. In the calibration context this will include both measurement error and model structural error. Let $\mathbf{k}$ denote the parameter set employed by the model. Let the matrix $\mathbf{X}$ represent the action of the (linearized) model. Then

$$\mathbf{h} = \mathbf{X}\mathbf{k} + \boldsymbol{\varepsilon} \tag{8.9.1}$$

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 variabilities (an expression of expert knowledge). Define the weight matrix $\mathbf{Q}$ such that

$$\mathbf{Q} = \mathbf{C}^{-1}(\boldsymbol{\varepsilon}) \tag{8.9.2}$$

where $\mathrm{C}(\boldsymbol{\varepsilon})$ is the covariance matrix of measurement noise.

Let the vector $\mathbf{j}$ represent a transformed parameter set, calculated from $\mathbf{k}$ using the equation

$$\mathbf{j} = \mathbf{F}^{-1/2}\mathbf{E}^{t}\mathbf{k} \tag{8.9.3}$$

so that

$$\mathbf{k} = \mathbf{E}\mathbf{F}^{1/2}\mathbf{j} \tag{8.9.4}$$

$\mathbf{E}$ and $\mathbf{F}$ are defined through singular value decomposition of the prior parameter covariance matrix $\mathrm{C}(\mathbf{k})$ (which also can be considered as the covariance matrix of innate parameter variability) through

$$\mathrm{C}(\mathbf{k}) = \mathbf{E}\mathbf{F}\mathbf{E}^{t} \tag{8.9.5}$$

Obviously

$$C(\mathbf{j}) = \mathbf{I} \tag{8.8.6}$$

Hence the transformed parameters **j** are normalized with respect to their innate variabilities. That is, they have been subjected to so-called Kahunen-Loève (KL) transformation. Where normalization of model outputs and model parameters is undertaken in the manner described above, equation 8.9.1 can be written as

$$\mathbf{f} = \mathbf{Zj} + \boldsymbol{\tau} \tag{8.9.7}$$

where

$$\mathbf{Z} = \mathbf{Q}^{1/2}\mathbf{XEF}^{1/2} \tag{8.9.8}$$

$$\mathbf{f} = \mathbf{Q}^{1/2}\mathbf{h} \tag{8.9.9}$$

and

$$\boldsymbol{\tau} = \mathbf{Q}^{1/2}\boldsymbol{\varepsilon} \tag{8.9.10}$$

Obviously

$$C(\boldsymbol{\tau}) = \mathbf{I} \tag{8.9.11}$$

SSSTAT bases its calculation of inversion statistics on equation 8.9.7.

SSSTAT works at two levels, the hiatus between these levels being marked by whether or not singular value decomposition of the **Z** matrix is undertaken. Singular value decomposition of **Z** leads to calculation of matrices **U**, **S** and $\mathbf{V}^t$ defined through

$$\mathbf{Z} = \mathbf{USV}^t \tag{8.9.12}$$

Prior to undertaking singular value decomposition 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 8.9.12 it follows that

$$\mathbf{ZZ}^t = \mathbf{US}^2\mathbf{U}^t \tag{8.9.13}$$

and that

$$\mathbf{Z}^t\mathbf{Z} = \mathbf{VS}^2\mathbf{V}^t \tag{8.9.14}$$

Note also that if measurement noise is ignored, then, from (8.9.7)

$$C(\mathbf{f}) = \mathbf{Z}C(\mathbf{j})\mathbf{Z}^t = \mathbf{ZZ}^t \tag{8.9.15}$$

Hence $\mathbf{ZZ}^t$ is the covariance matrix that denotes the variability of normalized model outputs **f** as this arises from the natural variability of normalized model parameters **j**. This can be compared with the variability of **f** 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 $\mathbf{V}$ based on removal of columns of $\mathbf{V}$ that correspond to non-zero (or non-near-zero) singular values. The vectors comprising the columns of $\mathbf{V}_1$ thus span the calibration solution space. Those comprising the orthogonal complement of $\mathbf{V}_1$, i.e. $\mathbf{V}_2$, span the calibration null space. Thus

$$\mathbf{V} = [\mathbf{V}_1 \ \mathbf{V}_2] \tag{8.9.16}$$

The matrix $\mathbf{V}_1\mathbf{V}^t_1$ denotes an orthogonal projection operator 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) and Doherty (2015) (which are also computed by the IDENTPAR utility). 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 square of the cosine of the angle between it and its projection onto the solution space.

Based on these considerations, as well as the characterisation of $\mathbf{Z}^t\mathbf{Z}$ provided through equation 8.9.14, each diagonal element of the $\mathbf{Z}^t\mathbf{Z}$ matrix can be viewed as the weighted squared cosine between the real-world (but unknown) scaled parameter vector $\mathbf{j}$ and its projection onto each of the vectors $\mathbf{v}_i$ comprising the columns of $\mathbf{V}$. 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. Zero values of s_i indicate no solution space projection; the parameter therefore projects entirely into the calibration null space. Nothing is then known of this parameter following the inversion process – not even its projection into the calibration solution space.

Where inversion is undertaken through singular value decomposition, the calibrated scaled parameter set $\underline{\mathbf{j}}$ is calculated through the equation

$$\underline{\mathbf{j}} = \mathbf{V}_1\mathbf{S}^{-1}_1\mathbf{U}^t\mathbf{f} = \mathbf{G}\mathbf{f} \tag{8.9.17}$$

where $\mathbf{G}$ is defined by the above equation. Doherty (2015) shows 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 (8.9.17) and (8.9.7) that

$$\underline{\mathbf{j}} = \mathbf{V}_1\mathbf{V}^t_1\mathbf{j} + \mathbf{V}^t_1\mathbf{S}^{-1}_1\mathbf{U}^t\boldsymbol{\tau} \tag{8.9.18}$$

so that

$$\underline{\mathbf{j}} - \mathbf{j} = -\mathbf{V}_2\mathbf{V}^t_2\mathbf{j} + \mathbf{V}^t_1\mathbf{S}^{-1}_1\mathbf{U}^t\boldsymbol{\tau} \tag{8.9.19}$$

Recalling that $C(\mathbf{j})$ and $C(\boldsymbol{\tau})$ are both identity matrices, it follows that

$$C(\underline{\mathbf{j}} - \mathbf{j}) = \mathbf{V}_2\mathbf{V}^t_2 + \mathbf{V}_1\mathbf{S}^{-2}_1\mathbf{V}^t_1 \tag{8.9.20}$$

After computing $\mathbf{ZZ}^t$ and $\mathbf{Z}^t\mathbf{Z}$, SSSTAT undertakes singular value decomposition of $\mathbf{Z}$. 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 $\mathbf{V}$ 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 singular value decomposition of the $\mathbf{Z}$ 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 $\mathbf{j}$. The latter complement these; these are the squares of the cosines of the angles between scaled parameters and their projections onto the calibration null space; $\mathbf{V}^t_1\mathbf{V}_1$ and $\mathbf{V}^t_2\mathbf{V}_2$ sum to $\mathbf{I}$. 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 Doherty (2015).

SSSTAT next computes and lists the diagonal elements of $\mathbf{V}_1\mathbf{S}^2_1\mathbf{V}^t_1$ and $\mathbf{V}_1\mathbf{S}^{-2}_1\mathbf{V}^t_1$. 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 are equal to or less than these. If the solution space comprises the entirety of parameter space, $\mathbf{V}_1\mathbf{S}^{-2}_1\mathbf{V}^t_1$ is equal to $(\mathbf{Z}^t\mathbf{Z})^{-1}$. If not, it is equal to the solution space component of the covariance matrix of post-calibration scaled parameter error. Each diagonal element of this matrix is thus the post-calibration solution space contribution to the error variance of the scaled parameter comprising the respective element of the vector $\mathbf{j}$. For each scaled parameter the sum of this diagonal element and the respective diagonal element of $\mathbf{V}_2\mathbf{V}^t_2$ is the total post-calibration error variance. SSSTAT lists these for all scaled parameters.

Finally, SSSTAT computes the diagonal elements of the matrix $\mathbf{U}_1\mathbf{S}^2_1\mathbf{U}^t_1$. Where all non-zero singular values are retained in the inversion process (which occurs if the inverse problem is well posed), the $\mathbf{U}_1\mathbf{S}^2_1\mathbf{U}^t_1$ matrix is equal to the $\mathbf{ZZ}^t$ matrix. If not, the diagonal elements of $\mathbf{U}_1\mathbf{S}^2_1\mathbf{U}^t_1$ 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 $\mathbf{U}_1\mathbf{S}^2_1\mathbf{U}^t_1$ are the error variances of respective model outputs arising from solution space scaled parameter variabilities.

8.9.3 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 from any observations covariance matrices supplied in the PEST control file. Relativity of these is preserved, while a factor is applied to all of them such that a new objective function is calculated from that supplied by the user; this new objective function is equal to the sum of non-zero-weighted observations comprising the calibration dataset. This is approximately its expected value if the weight matrix **Q** is equal to C^{-1}(**ε**).

It is important to note that any regularisation observations or prior information equations appearing in the PEST control file are ignored by SSSTAT. Hence the analysis carried out by SSSTAT is based purely on subspace concepts.

SSSTAT can obtain C(**k**) from a parameter uncertainty file (see section 2.5 of this manual). 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 records its calculated statistics (mainly the diagonal elements of matrices discussed above) to a file of the user's choosing. It also records the **Z** matrix of equation 8.9.7 and the **G** matrix of equation 8.9.17 in separate files. Each row of the former matrix 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 matrix 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. MATROW outputs can be listed in column format for greater readability using the MATTRANS utility. To make a bar chart based on this file, 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. If C(**k**) is non-diagonal then at least some elements of **j** will actually be linear combinations of elements of **k**. In spite of this, the same parameter names are used by SSSTAT for convenience.

8.9.4 Some Further Comments

SSSTAT provides a comprehensive set of statistics as they pertain to parameters and model

outputs employed in the ill-posed parameter estimation context. Further useful subspace information can be obtained using the SUPOBSPAR1 utility.

Finally, it is worth noting that formulation of an objective function and assignment of observation weights is as much of an art as it is a science. In formulating an objective function for use in the inversion process, some members of the calibration dataset may be processed in a number of different ways in formulation of the total objective function. As Doherty (2015) explains, this can provide some defense against the deleterious effects of model structural errors on the inversion process. 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, on the estimabilities of individual parameters, and on the information content of various observations or groups of observations.

8.10 OBSCOMP

8.10.1 General

OBSCOMP stands for "observation competition". The purpose of this utility program is to provide a modeller with some insights into why it is not possible to achieve a good fit with a particular measurement through the model calibration process. It is designed to answer questions such as "what other observations are "competing" with this observation, such that both of them cannot be fit at the same time?"

Let h_i represent the i'th observation of a calibration dataset. In search of a reason for failure of the calibration process to fit this observation, it is instructive to examine the parameters to which the observation is sensitive. A possible explanation for this observation not being well fit is that there are other observations that are sensitive to the same parameters as those to which it is sensitive. Perhaps the values of these other observations are irreconcilable with that of the observation in question because of the model's current construction and/or parameterization scheme. Model-to-measurement misfit may thus suggest a model defect. Or it may suggest that this measurement, or another competing measurement, is erroneous. Alternatively, if two or more observations are pulling the same parameters in the same direction, and a good fit with none of these observations has been attained through the inversion process, this suggests the existence of a more pervasive model defect that compromises the integrity of a number of allied model outputs.

Let $\mathbf{u}_i$ represent the row of the Jacobian matrix that corresponds to observation i; that is, $\mathbf{u}_i$ is the vector that comprises the i'th row of the Jacobian matrix. Let $\mathbf{u}_j$ represent the j'th row of the Jacobian matrix. We will (somewhat arbitrarily) define the "index of competition" between observation i and observation j as:

$$c_{ij} = (w_j w_i)\ \mathbf{u}^t_j \mathbf{u}_i \qquad (8.10.1)$$

where w_i and w_j are the weights associated with observations i and j. If c_{ij} is positive, then observations i and j are "pulling in roughly the same direction" on the parameters of the model. This means that if the residuals associated with these observations, namely r_i and r_j, are of opposite sign, then the calibration process has found a set of parameters that constitute a compromise between the values of these two observations, because it cannot satisfy both of them at the same time. This implies that uncorrelated measurement or structural noise is associated with one or both of these observations. If the noise is structural, this implies that

the model does not possess enough parameterisation detail to fit both of these observations at the same time; hence one or both of them may be the outcomes of local processes that are not well represented by the model.

In contrast, if c_{ij} is positive and if r_i and r_j are of the same sign, then the model is incapable of fitting either one of these observations, despite the fact that they are both pulling parameters in the same direction. This may indicate a high degree of noise associated with these observations and that a third observation is pulling parameters in the same direction; supposedly the residual associated with the third observation has the opposite sign to that associated with the first two observations. Alternatively, it may indicate a profound model structural defect whereby neither of these observations can be fit, in spite of their joint "pulling power" on model parameters.

Opposite considerations to the above apply where a c_{ij} is negative. In this case residuals of the same sign indicate high measurement noise or an inability of the model to fit measurement details. However if residuals have the opposite sign, this suggests competition with a third observation, or a more pervasive model structural defect.

8.10.2 Using OBSCOMP

OBSCOMP receives its user input in response to prompts that it writes to the screen. Its first prompt is:

```
Enter name of PEST control file:
```

Provide the name of a PEST control file, or simply the filename base of a PEST control file, in response to this prompt. Once it has received this name, OBSCOMP reads the PEST control file, as well as its associated Jacobian matrix file (which has the same filename base as the PEST control file, but possesses an extension of "*.jco*".) If the Jacobian matrix file does not exist, OBSCOMP ceases execution with an appropriate error message.

OBSCOMP's next task is to obtain the values of residuals attained through the inversion process. It asks the user where it can find these. Its prompt is:

```
Read residuals from REC, RES or REI file? [c/s/i]:
```

Suppose that the filename base of the PEST control file on which the inversion process was based is *case.pst*. Then the above files are named *case.rec*, *case.res* and *case.rei*. The first is the run record file, the second is the residuals file, while the third is the intermediate residuals file.

Once it has read the appropriate one of these files, OBSCOMP prompts for the name of an observation. This is the observation for which competing observations are sought. The prompt is:

```
Enter name of observation of interest:
```

This observation must be present within the PEST control file; additionally, it must have a weight which is greater than zero. Once it is in possession of this information, OBSCOMP asks for the name of the file in which it should record competition indices.

```
Enter name for observation competition index file:
```

The first and last parts of a competition index file are depicted in figure 8.4. The file begins with the name of the observation of interest, together with its residual. Then follows a table in which observations other than this are ordered according to index of competition with the observation of interest, from lowest to highest; residuals comprise the third column of this

file. It is the absolute value of the competition index that matters; hence a large negative competition index value is as significant as a large positive competition index value. The values (and the signs) of the residuals associated with competing observations are also important, as is discussed above. Note that if an observation has a weight of zero then, according to equation 8.10.1, its competition index with the observation of interest is zero.

```
 Observation of interest       = ar9
 Residual for this observation = 9.690410

 Observation            Competition_index    Residual
  ar2                    -1.910216          -0.1986810
  ar1                    -1.819081          -0.1240520
  ar3                    -1.656827          -0.1979860
  ar4                    -0.5974603         -6.6920000E-03
  ar19                     0.000000          2.5580000E-03
…
  ar11                     2.610246           9.479960
  ar6                      6.324030           1.755120
  ar10                     6.397111           10.90470
  ar7                      11.47613           3.895430
  ar8                      14.22743           6.849970
```

Figure 8.4 First and last parts of an observation competition index file.

OBSCOMP's final two prompts are:

```
 Enter name for sensitivities file:
 Enter no. of pos/neg competing observations to list sensitivities for:
```

The first column of the sensitivities file lists the name of every adjustable parameter featured in the inversion process (in the same order as in the PEST control file). The second column lists the weighted sensitivity of the observation of interest to every parameter. Suppose that the answer to the last of the above questions is *N*. Then the next *N* columns list the weighted sensitivities of the first *N* observations featured in the competition index file to these same parameters; these are the observations with the *N* lowest competition indices (i.e. the observations with the *N* most negative competition indices). The next *N* columns list the weighted sensitivities to all parameters of observations with the *N* highest competition indices. (Note that if an observation has a weight of zero, then its weighted sensitivies to all parameters is zero.)

9. Super Parameters and Super Observations

9.1 Introduction

This section describes a number of utilities which facilitate estimation and viewing of super parameters and super observations.

If the weighted Jacobian matrix $\mathbf{Q}^{1/2}\mathbf{J}$ is subjected to singular value decomposition we obtain

$$\mathbf{Q}^{1/2}\mathbf{J} = \mathbf{USV}^{t} \qquad (9.1.1)$$

Super observations are combinations of observations comprising the orthogonal columns of **U**. Super parameters are combinations of parameters comprising the orthogonal columns of **V**. As is described by Doherty (2015), **U** and **V** are orthonormal matrices. Each vector comprising a column of one of these matrices has a magnitude of unity and is orthogonal to other vectors comprising other columns of the same matrix. The i'th column of **V** (i.e. $\mathbf{v}_i$) is uniquely and entirely estimable from the i'th column $\mathbf{u}_i$ of **U**; the factor which links the two is the i'th singular value s_i comprising the i'th diagonal element of the matrix **S**. See section 6.2.6.3 of Doherty (2015) for further discussion.

The values of super parameters are estimated when PEST undertakes SVD-assisted inversion. The SVDAPREP utility constructs a PEST input dataset for use under these circumstances.

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. 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 the calibration solution space has 50 dimensions. Equation 6.2.48 of Doherty (2015) informs 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.

PEST memory requirements may become impossibly large where observations number in the tens or hundreds of thousands. A suitable strategy for approaching a parameter estimation problem such as this may be to 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 either standard or SVD-assisted parameter estimation. The

SUPOBSPREP utility constructs a PEST input dataset in which super observations replace normal observations.

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. Suppose, for example, 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 **U** is calculated on the basis of the weighted Jacobian matrix arising from this calibration problem, the first few columns of **U** 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 **U**. The "different aspects of the system" referred to above constitute different linear combinations of parameters which constitute the first few columns of **V**; coefficients of these columns can be estimated as so-called super parameters. Super observation/parameter pairs can be constructed using the SUPOBSPAR and SUPOBSPAR1 utilities.

PCLC2MAT computes only super parameters – the same super parameters that PEST estimates when undertaking SVD-assisted inversion. Changes in composition of super parameters as base parameters hit their bounds can be tracked using PCLC2MAT.

9.2 SVDAPREP

The SVDAPREP utility writes the PEST input dataset for SVD-assisted inversion. Recall from part I of this manual that the so-called "SVD-assist" methodology implemented by PEST supports estimation of so-called "super parameters". These are the scalar projections of real-world parameters onto a set of orthogonal vectors which collectively span the calibration solution space; see section 6.2.7 of Doherty (2015). As is described in part I of this manual, because PEST calculates sensitivities with respect to super parameters before calculating parameter upgrades, the computational savings achieved through SVD-assisted parameter estimation can be very high.

When using SVDAPREP to create a PEST input dataset for SVD-assisted inversion, and when implementing SVD-assisted inversion itself, super parameters are "invisible" to the user. Normally, all that matters is the progress of the inversion process itself, and the speed with which it can be implemented. However, if you are interested, the parameter composition of super parameters can be inspected using utility programs documented below.

SVDAPREP is not described in this part of the PEST manual. Instead it is extensively documented in part 1 of this manual where SVD-assisted inversion is described in detail.

9.3 PCLC2MAT

As described in part I of this manual, when PEST undertakes SVD-assisted parameter estimation, it estimates the values for a set of so-called "super parameters". These are usually named *par1* to *parN* where *N* is the total number of super parameters being estimated. These are back-transformed to base parameters before input files are written for the model on each occasion that the latter is run. The transformation process is undertaken by a utility named PARCALC which is run through the model batch file.

On each occasion that it runs the model, 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 N $\mathbf{v}_i$ vectors comprising the columns of the $\mathbf{V}$ matrix obtained through singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$. Each component of a particular $\mathbf{v}_i$ is actually the contribution that the respective base parameter makes to the respective super parameter. The ordering of base parameters in any $\mathbf{v}_i$ vector is the same as the ordering of parameters provided to the base PEST control file upon which the SVD-assisted parameter estimation process is based. The names of these base parameter are also listed in file *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 $\mathbf{v}_i$ vector recorded as a single column matrix.

As is apparent from the above command line syntax, the user-nominated column of the $\mathbf{V}$ matrix, i.e. the $\mathbf{v}_i$ vector, is written in PEST matrix file format; see section 2.4 of this manual for specifications of this format. The second part of the matrix file written by PCLC2MAT lists the names of base parameters (as matrix row names) while the first section provides the contribution made by each of these base parameters to the nominated $\mathbf{v}_i$ vector; the squares of these contributions sum to one as each $\mathbf{v}_i$ is a unit vector.

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 part I of this manual, the base parameter composition of super parameters may alter during the SVD-assisted inversion 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 inversion process and are therefore not included in the definition of any super parameters. Singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ is then repeated on the basis of the reduced number of adjustable base parameters; super parameters are 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 PCLC2MAT is run at the end of the parameter estimation process, then super parameter definition will correspond to the final super parameters employed by PEST. (If no base parameters have hit their bounds during this process then, of course, super parameter definition specified in *parcalc.in* will pertain to all base parameters.)

9.4 SUPOBSPREP

9.4.1 General

SUPOBSPREP builds a new PEST control file from an existing PEST control file in which super observations replace normal observations.

9.4.2 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 in part I of this manual, this can be easily produced by running PEST with NOPTMAX set to -1 or -2 in the "control data" section of the PEST control 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 in the new PEST input dataset whose task it is for SUPOBSPREP to build is a multi-step process. In the first of these steps the weight assigned to a super observation is equated to the singular value of $\mathbf{Q}^{1/2}\mathbf{J}$ 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 1E-6 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 have been assigned to super observations (together with an appropriate lower bound on these weights), the ratio of highest to lowest weight is 1E6. In many cases this range of weights is too large, as it 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 (see section 2.4 of this manual). $\mathbf{U}_1$ has as many columns as there are super observations, and as many rows as there are non-regularisation observations in the original PEST input dataset. Individual columns can be extracted from this file using matrix utilities such as MATCOLEX.

Finally SUPOBSPREP 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 adjustable parameters in the original PEST dataset. The columns of $\mathbf{V}_1$ provide the parameter combinations which the columns of $\mathbf{U}_1$ respectively inform.

9.4.3 What SUPOBSPREP Does

SUPOBSPREP 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 JCO file.
2. It forms the matrix $\mathbf{Q}^{1/2}\mathbf{J}$ where $\mathbf{Q}$ is the weight matrix associated with the current inverse problem and $\mathbf{J}$ 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 $\mathbf{Q}^{1/2}\mathbf{J}$ 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 file 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).

9.4.4 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 SVD-assist 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-calculates projected *residuals*, rather than projected observation *values*, 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.

9.4.5 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 case
```

at the command prompt, where *case* 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 inversion process can be eliminated by running PEST using the command

```
pest case /i
```

The "/i" switch is discussed in part I of this manual. When started with this switch, PEST prompts for the name of a JCO file from which it reads derivatives for use on the first 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 iteration. So if you wish to keep this file, copy it to another file before starting PEST.

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 PEST control file (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 value of 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.

9.4.6 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 you 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 then 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.

9.5 SUPOBSPAR

SUPOBSPAR makes calculation of super parameters and super observations 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 case N obsmatfile parmatfile
```

where

case	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 ignores regularisation prior information. (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; see section 2.4 of this manual for specifications of this 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) or 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 then, with a little cutting and pasting from other files, elements of super observations and/or super parameters can be plotted against space/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. Alternatively, bar charts of the observation/parameter content of super observations and super parameters can be plotted as histograms in software such as Microsoft EXCEL.

9.6 SUPOBSPAR1

9.6.1 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 Kahunen-Loève (KL) transformation to parameters prior to undertaking singular value decomposition on this matrix. The weighted Jacobian 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 super observations and super parameters have been calculated, SUPOBSPAR1 back-transforms the latter to native model parameters.

Theoretically, KL-transformation of parameters prior to their estimation ensures optimality of inversion undertaken though singular value decomposition as it leads to minimum error variance estimates of their values, and to minimum error variance predictions made by a calibrated model. Inversion based on KL-transformed parameters takes account of the information content of expert knowledge as encapsulated in a prior parameter covariance matrix. Super observations and super parameters calculated in this KL-transformed context therefore constitute the "natural" super observations and super parameters that best express information transfer from a calibration dataset to parameters on the one hand, and the blending of that information with expert knowledge on the other hand.

9.6.2 Theory

Let $C(\mathbf{k})$ represent the covariance matrix associated with the prior probability distribution of a parameter set $\mathbf{k}$. Through singular value decomposition, the $\mathbf{E}$ and $\mathbf{F}$ matrices (the latter being diagonal) can be defined through the equation

$$C(\mathbf{k}) = \mathbf{EFE}^t \tag{9.6.1}$$

The KL-transformed parameter set $\mathbf{j}$ is defined through

$$\mathbf{j} = \mathbf{F}^{-1/2}\mathbf{E}^t\mathbf{k} \tag{9.6.2}$$

so that

$$\mathbf{k} = \mathbf{E}\mathbf{F}^{1/2}\mathbf{j} \qquad (9.6.3)$$

Applying the classical formula for propagation of covariance readily shows that

$$C(\mathbf{j}) = \mathbf{I} \qquad (9.6.4)$$

Let the action of a linear model be represented by the matrix **Z**; where the model is nonlinear this is replaced by the Jacobian matrix **J** arising from local linearization of the model. Let observations comprising the calibration dataset be represented by **h**, and let **ε** represent measurement noise. Then

$$\mathbf{h} = \mathbf{Z}\mathbf{k} + \boldsymbol{\varepsilon} \qquad (9.6.5)$$

From the above it follows that

$$\mathbf{h} = \mathbf{Z}\mathbf{E}\mathbf{F}^{1/2}\mathbf{j} + \boldsymbol{\varepsilon} \qquad (9.6.6)$$

Let a weight matrix **Q** be selected such that

$$\mathbf{Q} = C^{-1}(\boldsymbol{\varepsilon}) \qquad (9.6.7)$$

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 of predictions which are sensitive to them. Pre-multiplying the model equation by $\mathbf{Q}^{1/2}$ we obtain

$$\mathbf{Q}^{1/2}\mathbf{h} = \mathbf{Q}^{1/2}\mathbf{Z}\mathbf{E}\mathbf{F}^{1/2}\mathbf{j} + \boldsymbol{\eta} \qquad (9.6.8)$$

where

$$\boldsymbol{\eta} = \mathbf{Q}^{1/2}\boldsymbol{\varepsilon} \qquad (9.6.9)$$

Obviously

$$C(\boldsymbol{\eta}) = \mathbf{I} \qquad (9.6.10)$$

In most circumstances of practical interest **Q** is a diagonal matrix, with elements equal to the squares of weights supplied in the PEST control file. Now let us perform singular value decomposition on the modified model matrix such that

$$\mathbf{Q}^{1/2}\mathbf{Z}\mathbf{E}\mathbf{F}^{1/2} = \mathbf{U}\mathbf{S}\mathbf{V}^t \qquad (9.6.11)$$

If measurement noise is ignored, it follows from equation 9.6.2 that

$$\mathbf{Q}^{1/2}\mathbf{h} = \mathbf{U}\mathbf{S}\mathbf{V}^t\mathbf{F}^{-1/2}\mathbf{E}^t\mathbf{k} = \mathbf{U}\mathbf{S}\mathbf{W}^t\mathbf{k} \qquad (9.6.12)$$

where **W** is defined by the above equation. "Super observations" computed by SUPOBSPAR1 are the columns of **U**, while super parameters computed by SUPOBSPAR1 are the rows of $\mathbf{W}^t$. Each of the former directly and completely inform each of the latter.

9.6.3 Running SUPOBSPAR1

SUPOBSPAR1 is run using the command

```
supobspar1 case 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 defining 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.

10. Linear Error and Uncertainty – Part I

10.1 Introduction

This chapter documents a suite of programs that form a basis for parameter and predictive error and uncertainty analysis, as well as ancillary analyses such as assessment of data worth and quantification of bias introduced to a defective model through the calibration process. At the heart of these programs are the PREDVAR utility suite and the PREDUNC utility suite "PREDVAR" stands for predictive error variance while "PREDUNC" stands for "predictive uncertainty. The GENLINPRED utility, which runs programs of the PREDVAR and PREDUNC suites automatically, is also documented in this chapter.

The chapter following this one also discusses utility programs that can be used for parameter error variance analysis. In general, the programs discussed in the present chapter should be used instead of those discussed in the next chapter as they are newer, more general, more flexible, and provide a greater a greater range of options.

10.2 SCALEPAR

10.2.1 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.

SCALEPAR re-formulates an inverse 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. Note that where a parameter is log-transformed in the inversion process, SCALEPAR takes this into account.

Once it has re-formulated the inverse problem, SCALEPAR writes a complete PEST input dataset for the new problem. In the new PEST control file 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, you should ensure that initial parameter values are indeed "most likely" parameter values; that is, you should ensure that they are the "expected values" of these parameters 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 the parameter's initial value as 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; this is run as part of the model ahead of any other components of the model in the new PEST input dataset generated

by SCALEPAR. 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.

Optionally, SCALEPAR writes a Jacobian matrix file (i.e. JCO file) corresponding to the new PEST control file. Sensitivities which occupy the elements of the new Jacobian matrix are computed from sensitivities occupying corresponding elements of a Jacobian matrix corresponding to the original PEST control file (if such a Jacobian matrix file exists).

Another option offered by SCALEPAR is the writing of a new parameter uncertainty file (see section 2.5 of this manual for specifications of this type of file), this pertaining to scaled parameters. By definition, the standard deviation of each scaled parameter is one.

10.2.2 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. 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 section 2.5 of the present document. Prior parameter uncertainties (i.e. the contents of the C(**k**) matrix discussed extensively in Doherty 2015) must be supplied in this file. A prior uncertainty must be supplied 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(**k**) 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 you can respond simply by pressing the <Enter> key in each case to accept the SCALEPAR default. The prompts are as follows.

```
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 the PAR2PAR utility. 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) before writing 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 part I of this manual, as PEST carries out the parameter estimation process, it records currently estimated 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 modeller, 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 parameter values which are optimised up to the stage of the parameter estimation process at which it is written. 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, depending on its control settings, if PEST has completed the parameter estimation process (or is halted using the "stop with statistics" option), it may undertake a final model run employing optimised parameter values. If this is the case then, 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 (see section 2.4 for specifications of this

type of file) containing the 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 "null" 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 to examine predictive error 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.)

Next SCALEPAR asks

```
Write scaled uncertainty file? [y/n]: (<Enter> for "n"):
```

If the response to this question is "y", SCALEPAR prompts 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. However 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 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 provided in the parameter uncertainty file.

10.2.3 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 with 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.

10.2.4 Calculating the Resolution Matrix of Native Parameters

The contents of the present subsection may be of interest to some, but will not be of interest to most. It is included anyway.

Suppose that unscaled parameters cited in the original PEST control file are designated by the vector **k** and that their scaled counterparts are designated by the vector **j**. Thus

$$\mathbf{j} = \mathbf{S}\mathbf{k} \tag{10.2.1}$$

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 PREDVAR-suite programs) regularised inversion has been carried out, and that a resolution matrix **R** has been calculated linking estimated scaled parameters $\underline{\mathbf{j}}$ to their real-world (but unknown) counterparts **j**. Thus

$$\underline{\mathbf{j}} = \mathbf{R}\mathbf{j} \tag{10.2.2}$$

Using 10.2.1, together with the relationship

$$\mathbf{k} = \mathbf{S}^{-1}\mathbf{j} \tag{10.2.3}$$

derived from 10.2.1, it is possible to compute the relationship between estimated native model parameters and their real-world counterparts as

$$\underline{\mathbf{k}} = \mathbf{S}^{-1}\underline{\mathbf{j}} = \mathbf{S}^{-1}\mathbf{R}\mathbf{j} = \mathbf{S}^{-1}\mathbf{R}\mathbf{S}\mathbf{k} \tag{10.2.4}$$

Thus the resolution matrix $\mathbf{R}'$ linking $\underline{\mathbf{k}}$ to **k** is

$$\mathbf{R}' = \mathbf{S}^{-1}\mathbf{R}\mathbf{S} \tag{10.2.5}$$

As already stated, SCALEPAR records the **S** and $\mathbf{S}^{-1}$ matrices in matrix file format. The utility program MATPROD can then be employed to implement the matrix multiplications

depicted in equation 10.2.5 if desired.

10.3 PREDVAR1

10.3.1 General

PREDVAR1 calculates the post-calibration error variance of a prediction without requiring that a model be actually calibrated to do so. It employs a slightly modified form of equation 6.1.25 of Doherty (2015) under the assumption that a notional calibration exercise has been completed using singular value decomposition. The equation used by PREDVAR1 is presented below.

$$\sigma^2_{s-\underline{s}} = \mathbf{y}^t(\mathbf{I} - \mathbf{R})C(\mathbf{k})(\mathbf{I} - \mathbf{R})^t\mathbf{y} + \sigma^2_r\mathbf{y}^t\mathbf{G}\mathbf{G}^t\mathbf{y} \qquad (10.3.1)$$

Items appearing in equation 10.3.1 are as follows.

$\sigma^2_{s-\underline{s}}$	is the error variance of a prediction;
s	is the true value of that prediction;
$\underline{s}$	is the value of that prediction made by the calibrated model;
$\mathbf{y}$	is the sensitivity of the prediction to parameters $\mathbf{k}$ employed by the model;
$C(\mathbf{k})$	is the prior parameter covariance matrix;
σ^2_r	is the measurement reference variance (see below);
$\mathbf{G}$	is the matrix from which the calibrated parameter set is calculated from the measurement dataset (see below);
$\mathbf{R}$	is the resolution matrix calculated as $\mathbf{GZ}$ where $\mathbf{Z}$ represents the linearized action of the model under calibration conditions (represented by the Jacobian matrix); and
$\mathbf{I}$	is the identity matrix.

The measurement reference variance is a proportionality constant linking the weights matrix $\mathbf{Q}$ used in the inversion process with the covariance matrix of measurement noise $C(\boldsymbol{\varepsilon})$. The following equation is thus assumed to describe the relationship between weights and measurement noise.

$$C(\boldsymbol{\varepsilon}) = \sigma_r^2\mathbf{Q}^{-1} \qquad (10.3.2)$$

σ_r^2 is roughly equal to the (measurement) objective function divided by the number of non-zero-weighted, non-regularisation measurements featured in the inverse problem.

Where calibration is achieved using singular value decomposition, $\mathbf{R}$ and $\mathbf{G}$ are calculated as follows.

$$\mathbf{G} = \mathbf{V}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t{}_1\mathbf{Q}^{1/2}\mathbf{h} \qquad (10.3.3)$$

$$\mathbf{R} = \mathbf{V}_1\mathbf{V}^t{}_1 \qquad (10.3.4)$$

where matrices $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$ appearing in the above equations are obtained from singular value decomposition of the weighted "model matrix" $\mathbf{Q}^{1/2}\mathbf{Z}$ as

$$\mathbf{Q}^{1/2}\mathbf{Z} = \mathbf{U}\mathbf{S}\mathbf{V}^t \qquad (10.3.5)$$

(Actually PREDVAR1 undertakes singular value decomposition of $\mathbf{Z}^t\mathbf{Q}\mathbf{Z}$ rather than $\mathbf{Q}^{1/2}\mathbf{Z}$; however the outcomes of its calculations are the same.) The "1" subscript on the $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$

matrices featured in equation 10.3.3 and 10.3.4 signify truncation of the singular value decomposition process at a user-specified number of singular values.

The action of the linearized model under calibration conditions is represented by the following equation.

$$\mathbf{h} = \mathbf{Zk} + \boldsymbol{\varepsilon} \tag{10.3.6}$$

See Doherty (2015) for full details. As explained in that text, one of the charms of linear analysis is that it can represent the calibration and prediction processes, and errors/uncertainties associated with the outcomes of these processes, without actually having to undertake these processes. All that it requires are sensitivities of model outputs used in the calibration and prediction processes to parameters employed by the model, as well as the C(**k**) and C(**ε**) matrices cited above. Ideally, sensitivities should be calculated using calibrated parameters. However this is not essential. In fact if a model is truly linear, then sensitivities would not change with parameter value; hence parameter values used in calculation of sensitivities would not matter at all.

If a model has not been calibrated, then the error variance of a prediction made by this model is the same as the prior uncertainty of this prediction. It is given by

$$\sigma^2_s = \sigma^2_{\underline{s}\text{-}s} = \mathbf{y}^t C(\mathbf{k})\mathbf{y} \tag{10.3.7}$$

Uses of equation 10.3.1 include the following.

1. A rapid assessment can be made of the reduction in predictive error 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, the innate variability C(**k**) of model parameters as this reflects the heterogeneity of the system, and the number of singular values used in estimation of parameters. By comparing post-calibration predictive error variance with pre-calibration predictive error variance, the worth of the calibration process in reducing the potential for error of one or a number of model predictions can thereby be computed.
2. Because the calibration exercise undertaken 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 basis for 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 acquisition of the former results in a greater reduction in predictive error variance than acquisition of the latter. This type of analysis is facilitated using the PREDVAR5 utility. (See also the PREDUNC5 utility which uses a linearized form of Bayes equation to conduct a similar analysis.)
3. By varying the terms of C(**k**) in accordance with an improvement in direct knowledge of system properties that may be gained through direct measurement of these properties, an assessment of the worth of such measurements in reducing the uncertainties of one or more predictions can be made. Various data acquisition strategies of this type can then be ranked. At the same time these strategies can be compared with the benefits of acquiring further information on system states (see above) for use in a future calibration exercise.
4. By setting certain elements, or groups of elements, of C(**k**) to zero and by employing equation 10.3.1 and/or 10.3.7 in conjunction with this revised C(**k**), the contribution

to pre- and/or post-calibration error variance of a specific model prediction made by different parameter types can be estimated. Where certain types of parameters are such that their properties are not directly measurable, this may allow the user to determine the "irreducible level of uncertainty" associated with key model predictions. This type of analysis is expedited using the PREDVAR4 utility. (See also the PREDUNC4 utility which uses a linearized form of Bayes equation to conduct a similar analysis.)

5. By re-calculating error variance using equation 10.3.1 for different numbers of singular values at which the singular value decomposition process is truncated, the minimum in the curve of model predictive error variance versus number of singular values can be ascertained. This sets the optimal dimensionality of the calibration solution space. See section 6.2.5 of Doherty (2015) for further details of this kind of analysis.

As is discussed below, all of the programs of the PREDVAR suite obtain C(**k**) by reading a parameter uncertainty file; see section 2.5 of this manual for specifications of this type of file. The C(**ε**) matrix is calculated from weights and/or observation covariance matrices supplied in the PEST control file using equation 10.3.2. However you must supply the reference variance σ_r^2 yourself. A quick way to calculate this is to divide the (measurement) objective function by the number of non-zero-weighted, non-regularisation observations comprising a calibration dataset. Alternatively, use the PWTADJ2 utility to create a PEST control file in which weights have been adjusted on an observation group by observation group basis, in such a way as to achieve a σ_r^2 value of unity.

10.3.2 Using PREDVAR1

PREDVAR1 requires too many inputs for these to be supplied through its command line. So it prompts you for the information which it needs. It 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 (i.e. a JCO file) is available for this PEST control file. This can be produced by setting NOPTMAX to -2 in the "control data" section of a PEST control file and then running PEST. PREDVAR1 reads the JCO file to obtain the **Z** matrix featured in the above equations.

Next PREDVAR1 prompts

```
Enter observation reference variance:
```

PREDVAR1 computes C(**ε**) from observation weights and observation covariance matrices featured in the PEST control file using equation 10.3.2. In doing so it assumes that the inverse of each weight contained in the PEST control file is proportional to the standard deviation of measurement noise associated with the observation to which it is assigned. The proportionality constant is assumed to be the same for all weights, and equal to the square root of the reference variance depicted in equation 10.3.2. Thus the squares of weights are assumed to comprise the diagonal elements of the **Q** matrix of this equation. In addition to weights, the PEST control file can feature one of more observation covariance matrices. The same reference variance must apply to these as well; that is, the elements of these user-supplied observation covariance matrices are multiplied by σ_r^2 of equation 10.3.2 in calculating the appropriate submatrix of C(**ε**). As stated above, a suitable value for the reference variance is easily calculated by dividing the actual or anticipated value of the (measurement) objective function by the number of non-zero-weighted, non-regularisation

observations featured in the PEST control file.

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(**k**). The following should be noted.

- If a parameter is log-transformed in the PEST control file, the elements of C(**k**) 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 denoted 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(**k**) 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 2.4 of this manual) 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 equations 10.3.1 and 10.3.7. 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 weights 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 matrix file can be constructed for each such prediction using the JROW2VEC utility.

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

featured in the PEST control file. Be aware, however, that if there are more adjustable parameters than observations featured in the PEST control file, the number of non-zero-valued singular values of the weighted Jacobian matrix can only be 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-valued singular value; in fact it will cease execution with an error message under these circumstances. 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 10.3.7. As the number of singular values prior to truncation increases, the first term of equation 10.3.1 normally falls. In fact it will fall monotonically if C(**k**) is equal to $\sigma^2_k\mathbf{I}$ where **I** is the identity matrix and σ^2_k 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(**k**) contains variances of very different magnitude, and/or indicates a high level of pre-calibration 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 latter can be rectified through Kahunen-Loève transformation.

The second term of equation 10.3.1 rises as the number of singular values prior to truncation increases. At some number of pre-truncation singular values there is normally an optimum, where the total predictive error variance (sum of the first and second terms of equation 10.3.1) is minimised. See section 6.2.5 of Doherty (2015) for details.

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 error variance made by the first and second terms of equation 10.3.1 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 input to a plotting/graphing program which can display the dependence of predictive error variance (and its two components) on number of pre-truncation singular values.

PREDVAR1's second output file is similar to that produced by PEST when implementing singular value decomposition as a solution device for the inverse problem. It lists singular values and respective eigencomponents of the weighted Jacobian matrix (actually $\mathbf{Z}^t\mathbf{Q}\mathbf{Z}$ rather than $\mathbf{Q}^{1/2}\mathbf{Z}$, as stated above). 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 eigencomponent 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 the weighted Jacobian matrix 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 generated; however parameter uncertainties and predictive sensitivities can change between these two runs. Note also that there may be tiny differences between predictive error variances calculated on the basis of stored SVD data and those calculated internally because of the slight loss of precision incurred through ASCII file storage.

It is 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 the **Z** matrix featured in the above equations. Thus it is assumed that the only type or regularisation undertaken in solving the notional inverse problem implied by equation 10.3.1 is that pertaining to truncated singular value decomposition, with the truncation limit set at singular values specified by the user.

10.4 PREDVAR1A

10.4.1 General

The capabilities of PREDVAR1A are very similar to those of PREDVAR1. However there are a number of notable differences, these being as follows.

1. Unlike PREDVAR1, PREDVAR1A does not undertake singular value decomposition of the $\mathbf{Z}^t\mathbf{QZ}$ matrix. Rather, it undertakes singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{Z}$. In many calibration contexts this is a numerically swifter procedure.
2. PREDVAR1A does not read or write an SVD file.
3. PREDVAR1A calculates error variance terms for not just one, but for many predictions.

10.4.2 Using PREDVAR1A

PREDVAR1A prompts, together with typical responses, 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]: 3
.
.
Enter no. of singular values before truncation [<Enter> if no more]: 50
Enter no. of singular values before truncation [<Enter> if no more]: <Enter>
```

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
```

Figure 10.1 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 error variances for all nominated singular values are to be written for the prediction whose parameter sensitivities are contained in the first file cited on that 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.

10.5 PREDVAR1B

10.5.1 General

PREDVAR1B implements theory that is demonstrated in White et al (2014) and that is discussed in section 9.6 of Doherty (2015). In particular, it implements equation 9.6.6 of Doherty (2015).

PREDVAR1B was designed to examine the error that is introduced to model predictions through use of an imperfect model to make those predictions. As all models are imperfect, analysis of defect-induced predictive error potential is important. However a requirement for analysis of this type of error is that "the perfect model" exists alongside the imperfect model so that errors incurred through use of the latter in place of the former can be quantified. Of course, the perfect model can never exist. In practice, a more detailed model is used as a surrogate for the perfect model. The outcomes of calculations made by PREDVAR1B based on this surrogate model are therefore only indicative.

White et al (2014) and Doherty (2015) examine some repercussions of the use of a defective model in the environmental decision-making context. Some of the conclusions from their study (which can be verified through use of PREDVAR1B) are as follows.

- Even though outputs of an imperfect model may fit historical field data well as the imperfect model is calibrated, the calibration process does not, of itself, guarantee that predictions made by the calibrated imperfect model will be accurate. However the theory behind PREDVAR1B demonstrates that, for some predictions, model defects can indeed be "calibrated out". For other predictions, the calibration process can introduce substantial bias – bias that would not be present if the model had not been calibrated. This happens because some parameters must play surrogate roles in the calibration process as they compensate for the notional omission of some system parameters from the defective model in order to allow the latter model to replicate historical behaviour of the system which it is meant to simulate.
- Those predictions which are similar in character to measurements which comprise the calibration dataset tend to be those which are most immune from calibration-induced predictive bias.
- On the other hand, those predictions which are partly sensitive to null-space parameter combinations and partly sensitive to solution-space parameter combinations of the "reality model" tend to be those which are most at risk of incurring calibration-induced predictive bias.
- The potential for calibration-induced predictive bias can often be reduced by seeking a fit with the calibration dataset which is somewhat diminished with respect to that which would be sought on the basis of measurement noise alone.
- Calibration-induced predictive bias can also be reduced through formulation of an appropriate objective function which "filters out" those aspects of the information contained within the calibration dataset which would otherwise flow into imperfect information receptacles provided by a defective model.

PREDVAR1B allows these issues to be explored. Its operation is very similar to that of PREDVAR1A. Recall that PREDVAR1A computes the solution space and null space contributions to predictive error variance as a function of the number of singular values employed in the calibration process. The greater the number of singular values, the better is the fit between model outputs and field data. However beyond an optimum number of

singular values, predictive error variance rapidly rises. The optimum level of fit is that which reduces predictive error variance to a minimum.

PREDVAR1B adds a third term to the predictive error variance equation, this being model-simplification error. This includes that which exists by virtue of model defects alone, as well as that which remains (or is amplified) after the imperfect model is calibrated because of the compensatory roles that parameters adjusted through the imperfect model calibration process may play. Like PREDVAR1A, PREDVAR1B allows a modeller to locate the minimum of the predictive error variance curve, and hence the optimum fit between model outputs and field data that should be sought through the calibration process; for an imperfect model the optimal fit is prediction-specific. It should be noted, however, that with simplification-induced predictive error taken into account, the location of the predictive-error variance minimum may not be as easy to find as when a perfect model is calibrated. Under the latter circumstances, a discrete error variance minimum results from the interplay between the monotonically decreasing (with increasing level of fit) null space contribution to predictive error variance and the monotonically increasing (with increasing level of fit) solution space contribution to predictive error variance (assuming that parameters have undergone Kahunen-Loève transformation prior to estimation). In fact, an error variance minimum may not even exist for some predictions made by a defective model, for it is possible that, for some predictions, any history-matching can do more harm than good.

10.5.2 Theory

The theory on which PREDVAR1B is based in presented in section 9.6.1 of Doherty (2015). As stated above, PREDVAR1B implements equation 9.6.6 of that text. This equation has three terms on its right side. The first two terms are the same as those calculated by PREDVAR1 and PREDVAR1A. The third term quantifies the contribution to predictive error variance incurred through use of a calibrated, defective model. The total error variance is the sum of these three terms.

In normal modelling practice it is not possible to calculate the third term of equation 9.6.6. However in order to obtain some measure of the cost of model simplification, it may be useful to build a complex model – more complex than you would actually use in practice. A Jacobian matrix can then be calculated using this complex model in order to explore the repercussions of using a simplified form of that same model in which some parameters (e.g. boundary conditions, certain stresses, certain processes, and/or certain expressions of heterogeneity) are fixed at values that you may normally deem to be "relatively error free" during the calibration and predictive processes. This mimics what happens in real-world modelling practice, whereby a simplified numerical expression of reality is declared (without any analysis) to be "fit for purpose" despite the fact that many actual or implied parameters are fixed at certain values – values that may in fact be wrong. The set of parameters that remains adjustable comprises the "defective" model for the purpose of PREDVAR1B analysis.

As is described in documentation for PREDVAR1 and PREDVAR1A above, it is good practice to scale parameters by their innate variability (an even better practice is to subject adjustable parameters to Kahunen-Loève transformation) before subjecting the resulting **Z** matrix to singular value decomposition. The SCALEPAR utility can help in this regard.

10.5.3 Using PREDVAR1B

See documentation for PREDVAR1A above. PREDVAR1B's prompts and typical responses

follow.

```
Enter name of PEST control file: case.pst
Enter observation reference variance i.e. phi(nonreg)/nobs(nonreg): .14

Enter name of parameter uncertainty file: param.unc
Enter name of predictive sensitivity matrix list file: sen.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]: 40
Enter no. of singular values before truncation [<Enter> if no more]: 40
Enter no. of singular values before truncation [<Enter> if no more]: 60
Enter no. of singular values before truncation [<Enter> if no more]: <Enter>

- reading PEST control file temp.pst....
- file temp.pst read ok.

Parameter groups containing a non-zero number of non-tied and non-fixed
parameters will now be listed.

Identify simplicity-correction parameter groups.
Other groups are presumed to contain only calibration-adjustable parameters.

  Parameter group "ro"?  [y/n]: n
  Parameter group "ro_0"?  [y/n]: y

- reading Jacobian matrix file temp.jco....
- file temp.jco read ok.

- reading predictive sensitivity matrix file ar18.vec....
- file ar18.vec read ok.

- undertaking SVD on observation covariance matrixes...
- 1 covariance matrices decomposed.

- carrying out singular value decomposition of Q(1/2)X....

- reading parameter uncertainty file param.unc....
- parameter uncertainty file param.unc read ok.

- computing error variance terms for truncation at 0 singular values  ....
- computing first term ....
- computing second term ....
- computing third term ....

- computing error variance terms for truncation at 1 singular values  ....
- computing first term ....
- computing second term ....
- computing third term ....
  etc.
```

As is clear from the above, the user is asked to partition parameters into those that are "calibration-adjustable" and those that are "correction parameters". Sensitivities of model outputs to all of these are required. They must lie within a JCO file computed by PEST that matches the PEST control file whose name is supplied to PREDVAR1B. A set of sensitivities of a prediction to all parameters, both calibration-adjustable model parameters and simplicity-correction parameters, must also be supplied. If the prediction of interest is featured in the same PEST control file as that read by PREDVAR1B (as a dummy observation with a weight of zero), predictive sensitivities can be extracted from the JCO file which complements the PEST control file using the JROW2VEC utility. Otherwise they must be extracted from another JCO file which features the same calibration-adjustable and simplicity-correction

parameters. Alternatively, to examine the post-calibration error variance of a single parameter, supply a "predictive sensitivity" vector which is comprised of zeroes except for the single element that represents the parameter of interest.

Simplicity-correction parameters must belong to one or more discrete parameter groups which are featured in the PEST control file supplied to PREDVAR1B. A parameter group must not house both calibration-adjustable parameters and simplicity-correction parameters.

The following points are worth noting.

- Weights employed in the calibration dataset should reflect measurement/structural noise; that is, weights should be proportional to the inverse of the standard deviations of measurement/structural noise. PREDVAR1B calculates C(**ε**) from these weights (and/or covariance matrices associated with measurements), together with the user-supplied reference variance. If weights are the inverse of the standard deviations of measurement/structural noise, and/or observation covariance matrice(s) are supplied that are equal to C(**ε**), then the reference variance is unity.
- You should request that enough singular values be employed to allow predictive error variance terms, and total predictive error variance, to be plotted against enough singular values for the minimum in the predictive error variance curve to be identified. However the maximum number of singular value to use should be no greater than either the maximum number of adjustable parameters, or the maximum number of non-zero-weighted observations, whichever is smaller. PREDVAR1B will inform you if more singular values than this are requested.
- A singular value of zero corresponds to no calibration at all.

10.6 PREDVAR1C

10.6.1 General

The PREDVAR1C utility is similar to the PREDVAR1B utility. However its functionality differs from that of PREDVAR1B in two important respects. These are as follows.

- PREDVAR1C insists that the prior covariance matrix C(**k**) be diagonal. Hence no parameters must show prior correlation.
- PREDVAR1C provides a measure of expected goodness-of-fit between model outputs and field data, this being the sum of the diagonal elements of the weighted residuals covariance matrix divided by the number of observations. Ideally, if each measurement weight supplied in the PEST control file is equal to the inverse of the standard deviation of respective measurement/structural noise, and/or observation covariance matrices assigned to observation groups in the PEST control file are equal to that of measurement/structural noise, then the sum of the diagonal elements of the weighted residuals covariance matrix should approximate the number of non-zero-weighted observations comprising the calibration dataset. This sum divided by the number of non-zero-weighted observations should therefore be approximately equal to 1.0. (Actually, it will be slightly above this value as the deployment of regularisation induces some degree of model-to-measurement misfit; see section 9.3.3 of Doherty, 2015).

10.6.2 Theory

The theory implemented by PREDVAR1C is discussed in section 9.3.2 of Doherty (2015).

However it is repeated here for clarity. We commence with equation 9.3.4 of Doherty (2015):

$$\mathbf{h} = \mathbf{Z}_m\mathbf{k}_m + \mathbf{Z}_d\mathbf{k}_d + \boldsymbol{\varepsilon} \quad (10.6.1)$$

In this equation $\mathbf{k}_m$ represents parameters employed by a simplified model whereas $\mathbf{k}_d$ represents "correction parameters" which encapsulate differences between a simple model and the complexity of the real-world (the latter being represented by a more complex model). $\mathbf{Z}_m$ and $\mathbf{Z}_d$ represent processes which act on $\mathbf{k}_m$ and $\mathbf{k}_d$ respectively. As usual $\mathbf{h}$ represents measurements comprising the calibration dataset while $\boldsymbol{\varepsilon}$ represents noise associated with these measurements.

Using singular value decomposition, the parameter field $\underline{\mathbf{k}}_m$ assigned to the calibrated simple model is estimated as

$$\underline{\mathbf{k}}_m = \mathbf{V}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t{}_1\mathbf{h} \quad (10.6.2)$$

which, after equation 10.6.1 is substituted into equation 10.6.2 becomes (after some simplification)

$$\underline{\mathbf{k}}_m = \mathbf{V}^t\mathbf{V}^t{}_1\underline{\mathbf{k}}_m + \mathbf{V}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t{}_1\boldsymbol{\varepsilon} + \mathbf{V}_1\mathbf{S}^{-1}{}_1\mathbf{U}^t{}_1\mathbf{Z}_d\mathbf{k}_d \quad (10.6.3)$$

Residuals $\underline{\mathbf{r}}$ pertaining to the calibrated model are calculated as

$$\underline{\mathbf{r}} = \mathbf{h} - \mathbf{Z}_m\underline{\mathbf{k}}_m = \mathbf{Z}_m\mathbf{k}_m + \mathbf{Z}_d\mathbf{k}_d + \boldsymbol{\varepsilon} - \mathbf{Z}_m\underline{\mathbf{k}}_m \quad (10.6.4)$$

After some manipulation of the terms of this equation we obtain

$$\underline{\mathbf{r}} = \mathbf{U}_2\mathbf{S}_2\mathbf{V}^t{}_2\underline{\mathbf{k}}_m + \mathbf{U}_2\mathbf{U}^t{}_2\boldsymbol{\varepsilon} + \mathbf{U}_2\mathbf{U}^t{}_2\mathbf{Z}_d\mathbf{k}_d \quad (10.6.5)$$

The covariance matrix of residuals of the calibrated model can be formulated from the above equation as

$$C(\underline{\mathbf{r}}) = \mathbf{U}_2\mathbf{S}_2\mathbf{V}^t{}_2C(\mathbf{k}_m)\mathbf{V}_2\mathbf{S}_2\mathbf{U}^t{}_2 + \mathbf{U}_2\mathbf{U}^t{}_2C(\boldsymbol{\varepsilon})\mathbf{U}_2\mathbf{U}^t{}_2 + \mathbf{U}_2\mathbf{U}^t{}_2\mathbf{Z}_dC(\mathbf{k}_d)\mathbf{Z}^t{}_d\mathbf{U}_2\mathbf{U}^t{}_2 \quad (10.6.6)$$

Suppose that the following conditions are met.

$$C(\mathbf{k}) = \mathbf{I} \quad (10.6.7a)$$

$$C(\mathbf{k}_c) = \mathbf{I} \quad (10.6.7b)$$

$$C(\boldsymbol{\varepsilon}) = \mathbf{I} \quad (10.6.7c)$$

Then equation 10.6.6 becomes

$$C(\underline{\mathbf{r}}) = \mathbf{U}_2\mathbf{S}^2{}_2\mathbf{U}^t{}_2 + \mathbf{U}_2\mathbf{U}^t{}_2 + \mathbf{U}_2\mathbf{U}^t{}_2\mathbf{Z}_d\mathbf{Z}^t{}_d\mathbf{U}_2\mathbf{U}^t{}_2 \quad (10.6.8)$$

PREDVAR1C internally normalizes parameters (and alters sensitivities accordingly) so that conditions (10.6.7a) and (10.6.7b) are met.

For the sake of keeping the above equations simple, user-specified weights employed in the inversion process are not cited in them. Nevertheless, equation 10.6.7c is respected if the user supplies weights in the PEST control file which are equal to the inverse of the standard deviation of measurement/structural noise, and/or supplies a covariance matrix for some or all observation groups which is equal to $C(\boldsymbol{\varepsilon})$ for each group. (In practice proportionality, together with a reference variance for weights and observation covariance matrices, can replace equality when using PREDVAR1C.)

If the number of singular values is zero, then the model is not calibrated at all; that is, no history-matching takes place. Under these circumstances equation 10.6.8 describes model-to-measurement misfit resulting from an uncalibrated, defective model, including that induced by the presence of measurement noise.

The first and second terms of equation 10.6.8 fall with increasing number of singular values employed in the inversion process (the number of columns comprising $\mathbf{U}_2$ decreases with increasing singular value truncation point). When singular values fall to zero with increasing singular value index, so do contributions made by the first term. The second term is equal to the identity matrix **I** when zero singular values are used in model calibration (that is when the model is not calibrated at all); it falls uniformly as the number of singular values used in the inversion process increases.

The behaviour of the third term of 10.6.8 as singular values increase is context-specific. If $\mathbf{Z}_d$ is orthogonal to $\mathbf{U}_2$, then it is zero. When zero singular values are employed in the calibration process (i.e. when the model is not calibrated at all), this implies that model defects do not affect model outputs that correspond to members of the calibration dataset. Alternatively, this term may start off as non-zero, but fall to a low number as the number of singular values employed in the calibration process increases. This implies that model parameters that are adjusted through the calibration process (i.e. the $\mathbf{k}_m$ parameters as distinct from the $\mathbf{k}_d$ parameters) can assume values that compensate for model defects (i.e. for the presence of $\mathbf{k}_d$ parameters) to ensure that the calibration process achieves a good fit with the calibration dataset.

If a model has no defects then the third term of equation 10.6.8 is absent. In this case, if the conditions implied by equations 10.6.7a and 10.6.7c are respected, the diagonal elements of C(**r**) should be in the vicinity of 1.0 as the number of singular values employed in the inversion process increases to the point at which the model can be considered as "calibrated". This is the point at which parameter and predictive error variances are minimized – see documentation of PREDVAR1 and PREDVAR1A. The second term of equation 10.6.8 will be less than 1.0 at this point, while the first term will have descended from possibly high values to values of 1.0 or below. (Eventually this term will be zero; this occurs when the singular value truncation point is such that singular values are zero.)

10.6.3 Using PREDVAR1C

The running of PREDVAR1C is identical to the running of PREDVAR1B except for the fact that PREDVAR1C asks two other questions. These are as follows.

```
Calculate residuals statistics? [y/n]:
Enter name for residuals statistics file:
```

The second question is asked only if the response to the first question is "y".

The first part of a residual statistics file written by PREDVAR1C is illustrated in Figure 10.2.

sing_val	first_term	second_term	third_term	sum_of_terms
0	1.105678	1.000000	0.7587583	2.864436
1	0.6062968	0.9473684	0.7545655	2.308231
2	0.2745901	0.8947368	0.7279764	1.897303
3	0.1040756	0.8421053	0.6870091	1.633190
4	3.3155639E-02	0.7894737	0.6434713	1.466101
5	8.8168476E-03	0.7368421	0.6037423	1.349401
6	1.9222139E-03	0.6842105	0.5711024	1.257235
7	4.0396902E-04	0.6315789	0.5486338	1.180617

Figure 10.2 The first part of a residual statistics file.

The order of terms referenced in the header to the residual statistics file is the same as that of terms featured in equation 10.6.8. The items listed in each column are the sum of the diagonal elements of C(**r**) calculated using only that term, with this sum divided by the number of non-zero-weighted observations featured in the PEST control file.

When using PREDVAR1C it is important to keep the following points in mind.

1. As has already been discussed, weights supplied in the PEST control file should be proportional to the inverse of the standard deviation of measurement/structural noise. This strategy allows condition 10.6.7c to be satisfied once the proportionality constant is supplied through the reference variance. Alternatively (or as well), a covariance matrix proportional to C(**ε**) can be supplied for one or more observation groups. (If desired, the PWTADJ2 utility followed by JCO2JCO can be used to create a PEST control file with correct weights together with a corresponding JCO file prior to running PREDVAR1C; the reference variance associated with the PWTADJ2-created PEST control file is 1.0.)
2. If the above strategy is followed then a "sum_of_terms" value (i.e. final column of the residuals file) approaching 1.0 indicates a fit with the field data that is commensurate with that expected from measurement noise alone.
3. Reduction of the value of the third term with an increasing number of singular values indicates that model defects are being "calibrated out". Adjustable parameters may therefore be adopting surrogate roles to achieve a high level of model-to-measurement fit.
4. PREDVAR1C writes all output files that are written by PREDVAR1B. Hence it can be used as a replacement for PREDVAR1B. In fact, its use is preferred to that of PREDVAR1B as the internal normalization of sensitivities to achieve condition 10.6.7a results in a monotonic relationship between the null space component of predictive error variance and number of singular values used in the inversion process.
5. Regularisation observations and prior information equations cited in a PEST control file that is read by PREDVAR1C are ignored.

Calculations undertaken by PREDVAR1C are memory and cpu-intensive. Hence execution speed may be slow. If observation numbers are high, memory and cpu requirements may be impossible to meet. If this occurs use a 64 bit version of this utility. Consider also reducing the number of observations featured in the PEST control file.

10.7 PREDVAR2

10.7.1 General

In contrast to PREDVAR1B and PREDVAR1C, and in similar fashion to other members of the PREDVAR suite, PREDVAR2 does not consider model defects. Operation of PREDVAR2 is, in fact, based on the same principles as those on which PREDVAR1is based. However instead of calculating the dependence of predictive error variance on the number of singular values employed in the truncated singular value decomposition inversion process through which **R** and **G** of equation 10.3.1 are evaluated, it supports calculation of predictive error variance based on a sequence of different parameter uncertainty files, all for the same singular value truncation limit. This can serve 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 can be construed as 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 is 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" parameter 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.

Like PREDVAR1 (but unlike PREDVAR1A), PREDVAR2 undertakes singular value decomposition of the $\mathbf{Z}^t\mathbf{Q}\mathbf{Z}$ matrix rather than the $\mathbf{Q}^{1/2}\mathbf{Z}$ matrix (where $\mathbf{Z}$ is defined in equation 10.3.6). Hence its execution may be a little slow where parameter and/or observation numbers are large.

Note however that PREDUNC-suite programs documented later in this section often provide better measures of data worth than PREDVAR-suite programs as PREDUNC-suite programs calculate parameter and predictive uncertainty rather than error variance. Uncertainty can be quicker to calculate; furthermore uncertainty provides a slightly more intrinsic measure of the information content of a calibration dataset than does error variance, the latter being a function of the regularisation device employed to achieve calibration uniqueness, in this case singular value decomposition at a user-specified truncation point.

10.7.2 Using PREDVAR2

Like PREDVAR1, PREDVAR2 commences execution with the prompt

```
Enter name of PEST control file:
```

It is assumed that a JCO file exists corresponding to the PEST control file whose name is provided in response to the previous prompt. (PREDVAR2 will cease execution with an error message if this is not the case). 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 comprised of 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 a PEST control file in which these parameter values feature as initial values; NOPTMAX should be set to -2 in this file. In other circumstances it may be convenient to build this JCO file from a JCO file recorded 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 error variances, and that any observation covariance matrices cited in the PEST control file are related to true measurement error covariances using the same proportionality constant. PREDVAR2 asks for the value of this constant, that is for the reference variance. The prompt is

```
Enter observation reference variance:
```

A suitable value for the reference variance can be obtained by dividing the (measurement) objective function by the number of non-zero-weighted, non-regularisation observations

featured in the inverse problem.

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
```

Figure 10.3 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). Specifications for a parameter uncertainty file are provided in section 2.5 of this manual. There is no limit to the number of parameter uncertainty files which can be so listed. Each one of these files allows construction of a C(**k**) matrix for use in equation 10.3.1.

Next PREDVAR2 asks for the name of a predictive sensitivity matrix file. This contains the **y** vector featured in equation 10.3.1. This matrix can be built using a text editor if desired (see the format for this file in section 2.4 of this manual). 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 zero 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 output file will contain predictive error variance components corresponding to the first and second terms of equation 10.3.1, 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 $\mathbf{Z}^t\mathbf{QZ}$ 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 which define orthogonal directions in parameter space which are used as a basis for subdividing that space into calibration solution and null subspaces). However if this file is read in a subsequent PREDVAR2 run (instead of undertaking singular value decomposition of $\mathbf{Z}^t\mathbf{QZ}$ again), considerable savings in computation time can be achieved where large numbers of adjustable parameters are cited in the PEST control file (with slight loss of precision incurred 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 run. If you press 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.

10.8 PREDVAR3

10.8.1 General

PREDVAR3 bears 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. It has been largely superseded by PREDVAR4 however (see below). Hence it is recommended that the latter program be used instead of PREDVAR3. Nevertheless PREDVAR3 is retained in the PEST utility suite as it may prove useful to some.

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, 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 innate variabilities) to determine the minimum predictive error variance achievable for a certain prediction. This is accomplished by plotting predictive error variance versus number of singular values, and identifying the minimum of this curve.
2. Build a new PEST control file 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 minimised 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 these parameters to the error variance of the prediction of interest.

PREDVAR3 carries out the same operations as listed above, but without the need for

production of new PEST control file and a new Jacobian matrix file, and without the need to run PREDVAR1 repeatedly. Because of its similarity to PREDVAR1, many of its inputs resemble that of this program.

10.8.2 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 observation covariance matrices are supplied whose elements are related to measurement error variance 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.

```
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 you will know roughly where the minimum will lie, and can properly span this singular value interval. If the interval 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 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 to its output file the singular value corresponding to the minimum predictive error variance for each parameter type; see below.

Next PREDVAR3 prompts

```
Enter name of parameter type list file:
```

The format of such a file is illustrated in figure 10.4.

```
* 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.
```

Figure 10.4 Example 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 adjustable 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 list 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(**k**) matrix for the current inverse problem indicates that correlation exists between certain parameters (that is, if C(**k**) is not a diagonal matrix), then correlated parameters must belong to the same parameter type, and must 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(**k**) matrix be conditioned as a result of the attainment of perfect knowledge of the values of the notionally fixed parameters. Where a fixed parameter is not correlated with any other parameter, such conditioning of C(**k**) 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
```

Figure 10.5 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.

10.9 PREDVAR4

PREDVAR4's purpose and use are identical to those of PREDVAR3. However it offers a slightly higher level of functionality than that offered by PREDVAR3.

If certain parameters are frozen (i.e. if perfect knowledge of the values of these parameters is assumed) for the purpose of calculating their contribution to predictive error variance, and if these parameters are correlated with non-frozen parameters, the C(**k**) 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(**k**) matrix in predictive error variance computation results in an altered error variance for many predictions, this reflecting the fact that notional perfect knowledge of the frozen parameters has implications for knowledge of correlated (from a prior knowledge point of view) non-frozen parameters as well.

Conditioning of the C(**k**) covariance matrix is carried out using equation 3.8.5 of Doherty (2015).

See also the PREDUNC4 utility. This computes the contributions of different parameter types to predictive uncertainty rather than to predictive error variance. This is quicker to compute; furthermore uncertainty is a more "natural" quantity than "error" when assessing parameter importance.

10.10 PREDVAR5

10.10.1 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 existing observations by calculating predictive error variance with selected ones, or groups, of these observations removed from the calibration dataset.
2. It allows ranking of the relative worth of new observations by calculating predictive error 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. These groupings are provided in an "observation type list file", the format of which is very similar to that of the parameter type list file used by PREDVAR3 and PREDVAR4. Figure 10.6 illustrates such a file.

```
* 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.
```

Figure 10.6 Format of an observation type list file.

The PREDVAR5 observation addition and subtraction options are now discussed in more detail.

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 affected by setting all of the pertinent observations weights to zero. Not only observations can be removed, but prior information equations can be removed 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 suite 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 inverse problem.

An observation type can have no members; often the first nominated observation type has 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.

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 inverse 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 inverse 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 regularisation 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 equation 10.3.1 (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 must be 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. You 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.

10.10.2 Using PREDVAR5

PREDVAR5 commences execution with the prompt

```
Enter name of PEST control file:
```

As for the other PREDVAR-suite programs, the PEST control file provides specifications for 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. a 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 -2 so that the run was carried out specifically for this purpose), or it may have been created using the JCO2JCO utility from an existing JCO file pertaining to another PEST control file citing a superset of the parameters and observations which comprise the current inverse problem.

Then it asks

```
Enter observation reference variance:
```

This is discussed below. PREDVAR5's next prompt is

```
Enter name of parameter uncertainty file:
```

This file provides the C(**k**) matrix of equation 10.3.1; see section 2.5 of this manual for its specifications. Next PREDVAR5 prompts

```
Enter name of predictive sensitivity matrix file:
```

The contents of this file must be in accordance with PEST matrix file format. It must contain 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, and is not fixed or tied, but is not cited 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 similar to its role for other PREDVAR-suite programs. In the present case it provides a list of singular values for which PREDVAR5 calculates predictive error variance for each of the various observation types for which this is required. It chooses 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, as must occur when predictive error variance differences are calculated.

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 you select 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 new observations (the latter can be computed using an empty observation type if you wish) 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 withdrawn to compute the effect that each observation type has in achieving the predictive error variance of the currently calibrated model.

10.10.3 Observation Weights and Reference Variance

Like other PREDVAR-suite programs, PREDVAR5 assumes that observation weights supplied in the PEST control file are proportional to the inverse of measurement noise variances (squares of standard deviations). An observation covariance matrix may be supplied for one or a number of observation groups cited in the PEST control file if you wish; the same constant of proportionality (i.e. the reference variance) is assumed to prevail between this matrix and measurement noise variance. Use of the PWTADJ2 utility following a calibration exercise can create a PEST control file in which this constant of proportionality is 1.0. Alternatively, a value for the reference variance can be estimated by dividing the calculated or estimated value of the (measurement) objective function associated with the calibration process by the number of non-zero-weighted, non-regularisation observations used in this process.

10.10.4 Parameter Scaling

Ideally, parameters should be scaled using the SCALEPAR utility before using any of the PREDVAR-suite programs (except PREDVAR1C). This helps to ensure 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(**k**) matrix). It also promotes minimisation 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.

10.10.5 PREDUNC5 instead of PREDUNC4

See the PREDUNC5 utility documented below. This uses predictive uncertainty rather than predictive error variance to assess observation worth. Predictive uncertainty is quicker to compute than predictive error variance. No parameter scaling is required. Predictive uncertainty does not suffer the same singular value granularity as does predictive error variance. In addition to this, uncertainty is a more "natural" quantity than "error" (though the two should differ only slightly – see Doherty (2015)). Hence use of PREDUNC5 is recommended over use of PREDVAR5 for assessment of data worth.

10.11 PARVAR1

PARVAR1 calculates the post-calibration error variance of all parameters employed by a model. Like PREDVAR-suite programs, it assumes that model calibration takes place through singular value decomposition; the user informs PARVAR1 of the dimensionality of the solution space (i.e. the number of pre-truncation singular values used in notional estimation of parameters). It uses the same formula to calculate parameter error variance as

that used by PREDVAR1A to calculate predictive error variance. However the "predictive sensitivity vector" of a parameter is simply a vector in which all elements of the vector are zero except for that pertaining to the parameter of interest; the value of this sensitivity is 1.0. The calculation is repeated for each adjustable parameter.

Typical prompts and responses are as follows.

```
Enter name of PEST control file: pestcase.pst
Enter observation reference variance: 2.5

Enter name of parameter uncertainty file: param.unc

Enter no. of singular values before truncation: 12
Enter name for output file: var.tab
```

Once PARVAR1 has been informed of the name of a PEST control file, it immediately checks that the corresponding Jacobian matrix file is also present; if this is not the case it ceases execution with an appropriate error message. It ignores all regularisation observations and prior information that is contained in the PEST control file when computing post-calibration parameter error variance.

As usual, the observation reference variance can be calculated as the actual or estimated (measurement) objective function divided by the number of non-zero-weighted, non-regularisation observations included in the calibration dataset.

The format of a parameter uncertainty file is specified in section 2.5 of this manual.

The first part of a PARVAR1 output file is shown below.

```
Parameter error variances for 12 singular values
 -------------------------------------------------
 parameter   variance_1     variance_2      total_variance    standard_deviation
 k_ppt1       0.5830897      1.1113309E-03     0.5842010        0.7643304
 k_ppt2       0.5905922      3.2496525E-04     0.5909172        0.7687114
 k_ppt3       0.5957456      1.2426367E-04     0.5958699        0.7719261
 k_ppt4       0.5886176      6.6431960E-04     0.5892819        0.7676470
 k_ppt5       0.5885078      6.6709287E-04     0.5891749        0.7675773
 k_ppt6       0.5956354      1.3614296E-04     0.5957715        0.7718624
 k_ppt7       0.5913795      2.6715074E-04     0.5916467        0.7691857
 k_ppt8       0.5817068      1.3035155E-03     0.5830103        0.7635511
 k_ppt9       0.5420429      7.3421960E-03     0.5493851        0.7412052
 k_ppt10      0.5616598      3.4346757E-03     0.5650944        0.7517276
 k_ppt11      0.5856337      7.0444812E-04     0.5863381        0.7657272
 etc.
```

Figure 10.7 Part of a PARVAR1 output file.

In the table that is recorded on the PARVAR1 output file, "variance_1" and "variance_2" refer to the null space and solution space contributions to total posterior parameter error variance; these are the first and second terms of equation 10.3.1. These sum to the total parameter error variance, the square root of which is the post-calibration parameter standard deviation; the last of these is tabulated in the final column of the PARVAR1 output file.

10.12 PREDUNC1

10.12.1 General

The PREDUNC suite of programs have much in common with the PREDVAR suite of programs. However their major difference is that they calculate predictive uncertainty variance rather than predictive error variance. Hence parameter importance (PREDUNC4)

and observation data worth (PREDUNC5) are assessed in terms of their effects on predictive uncertainty rather than on predictive error variance. (Note that the value of a parameter can be considered to be a model prediction; the sensitivity vector for this prediction has elements of zero, except for that pertaining to the parameter of interest, for which the sensitivity value is 1.0.)

The PREDUNC suite of programs base their analyses on equations 7.4.1 and 7.4.2 presented in Doherty (2015); these are the same equation (the linearized form of Bayes equation) but expressed in different ways. The assumptions that underpin derivation of these equations are as follows.

1. The model is linear, so that its action can be replaced by a Jacobian matrix (the **Z** matrix in the equations referred to above);
2. Prior parameter probabilities expressed by the covariance matrix C(**k**) are Gaussian;
3. Measurement noise expressed by the covariance matrix C(**ε**) is Gaussian.

PREDUNC1 is the simplest member of the PREDUNC suite. Its purpose is solely to calculate the pre- and post-calibration (i.e. prior and posterior) uncertainty variance of a prediction. (Recall that the variance is the square of the standard deviation.)

10.12.2 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 (i.e. the uncertainty variance of prediction *s*); hence singular values are not requested.

Like PREDVAR1, PREDUNC1 commences execution with the prompt

```
Enter name of PEST control file:
```

PREDUNC1 checks that a JCO file exists which complements the PEST control file. If it does not exist, PREDUNC1 ceases execution with an appropriate error message. PREDUNC1 (and other members of the PREDUNC suite) will also cease execution with an error message if the PESTMODE variable in the PEST control file is set to "regularisation". If your PEST file includes regularisation observations and/or prior information, these can be removed using the SUBREG1 utility. A complementary JCO file for this new PEST control file can then be built using the JCO2JCO utility.

PREDUNC1's next prompt is

```
Enter observation reference variance:
```

As is done by members of the PREDVAR utility suite, PREDUNC-suite programs obtain 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 the respective measurement, the proportionality constant being the reference variance σ^2_r. Thus the standard deviation of measurement noise associated with each measurement is presumed to be σ_r times the inverse of the weight ascribed to that measurement, where σ_r is the square root of the reference variance supplied by the user. Where an observation covariance matrix is supplied for some or all observations, this same reference variance is assumed to apply.

A value for the reference variance can be calculated by dividing the calculated or expected value of the objective function by the number of non-zero-weighted observations featured in the inverse problem.

PREDUNC1's next prompt is

```
Enter name of parameter uncertainty file:
```

in response to which the name of a file specifying prior parameter uncertainties (i.e. the C(**k**) matrix) must be supplied; see section 2.5 of this manual for specifications of this type of file. 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 (as a vector) the sensitivity of a prediction to each adjustable parameter. Note that the ordering of 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 optimised for small number of parameters" is equation 7.4.2 of Doherty (2015); the version optimised for small number of observations is equation 7.4.1. If in doubt chose the first of these alternatives as its calculation requires fewer matrix operations.

PREDUNC1 next computes the pre- and post-calibration uncertainty of the prediction, and writes these to the screen.

Unlike PREDVAR-suite utilities, PREDUNC-suite utilities do not compute 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 (or decreasing the value supplied for the reference variance). When measurement weights are very high (and hence measurement noise is assumed to be low), 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.

10.13 PREDUNC4

PREDUNC4 is to PREDUNC1 what PREDVAR4 is to PREDVAR1. That is, it computes the decrease in 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" (see documentation of PREDVAR4 for specifications of this file). This decrease in uncertainty accrued through acquiring perfect knowledge of the values of this group of parameters can be loosely described as the "contribution that this group of parameters makes to the uncertainty of the prediction". Unlike PREDVAR4, PREDUNC4 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 Jacobian matrix be undertaken.

Refer to documentation of PREDVAR4 for usage details of PREDUNC4. In general, it is better to use PREDUNC4 than PREDVAR4 for assessment of parameter importance with respect to a particular prediction, as uncertainty is an easier and "more natural" quantity to

compute than error.

10.14 PREDUNC5

PREDUNC5 is to PREDUNC1 what PREDVAR5 is to PREDVAR1. That is, it computes alterations to the uncertainty of a prediction where groups of observations are sequentially added to, or subtracted from, the calibration dataset. The names of these observations are supplied in an "observation type list file", specifications for which are supplied with documentation of PREDVAR5. Unlike PREDVAR5, PREDUNC5 does not prompt for the name of 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 Jacobian matrix be undertaken.

Refer to documentation of PREDVAR5 for usage details of PREDUNC5. In general, it is better to use PREDUNC5 than PREDVAR5 for assessment of data worth with respect to a particular prediction, as uncertainty is an easier and "more natural" quantity to compute than error.

10.15 PREDUNC6

PREDUNC6 performs a similar function to PREDUNC1 in that it computes pre- and post-calibration predictive uncertainty. Unlike PREDUNC4 and PREDUNC5 however, it does not compute any "value-added" information such as contributions made to predictive 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 when it commences execution; typical responses are also shown.

```
Enter name of PEST control file: case13.pst
Enter observation reference variance: 1.0

Enter name of parameter uncertainty file: param.unc
```

As usual, a JCO file must exist to complement the PEST control file. For the above example this file would be named *case13.jco*.

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 sensitivities of one or many "observations" which are 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 the inverse problem is based - that is the PEST control file whose name is supplied in response to the first of PREDUNC6's prompts. 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 of predictive sensitivity vectors.

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, together with 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. If in doubt enter "1".

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 which defines the inverse problem. 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.

10.16 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. The first is through equation 7.3.15 of Doherty (2015) while the second is through equation 7.3.4 of Doherty (2015). Both equations compute the same thing; however the matrix manipulations are somewhat different. In theory, use of the second equation is slow if there are many observations while use of the first equation is slow if there are many parameters. In practice, because equation 7.3.4 involves a large number of matrix multiplications, its use can incur numerical errors where parameter numbers are large. In either case, calculation of the posterior covariance matrix can take a long time where parameter numbers are very high.

PREDUNC7's prompts are as follows.

```
Enter name of PEST control file:
Enter observation reference variance:

Enter name of prior parameter uncertainty file:

Enter name for posterior parameter covariance matrix file:
Enter name for posterior parameter uncertainty 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:
```

Option 1 uses equation 7.3.15 while option 2 uses equation 7.3.4 from Doherty (2015). If in doubt use option 1 (because it is less susceptible to numerical error).

The covariance matrix computed by PREDUNC7 is written in PEST matrix file format to a file of your choice. Note that each row of the matrix wraps onto a new line after each succession of eight element values. The posterior parameter uncertainty file simply cites this matrix file within a COVARIANCE_MATRIX block. The posterior parameter uncertainty file can be used by programs such as RANDPAR to generate random samples from the posterior parameter probability distribution.

In calculating $\mathbf{C}(\boldsymbol{\varepsilon})$ 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. The reference variance can be approximated as the objective function achieved or expected through calibration divided by the sum of non-zero-weighted observations. (It is not wise to include prior information in the PEST control file provided to PREDUNC7 - or to any other member of the PREDUNC suite - as expert parameter knowledge is accounted for in the prior parameter uncertainty file.)

10.17 GENLINPRED

10.17.1 General

GENLINPRED stands for "generalized 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.

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 computational 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.

10.17.2 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 (and is far quicker to calculate than relative error variance reduction).

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 the estimated value of a parameter) at different singular value truncation levels. This allows you to graph the dependence of these quantities on the number of pre-truncation singular values employed in calibration of a model. It uses PREDVAR1A 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 (or 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 (or 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 (or 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 accrued.

All operations are carried out on a PEST input dataset supplied by the user. GENLINPRED internally modifies this dataset by scaling parameters by their pre-calibration variabilities. Where calibration is notionally, or actually, implemented using truncated singular value decomposition, this operation results in lower predictive error variance than where unscaled parameters are estimated through the inverse problem solution process.

10.17.3 Predictive Error Variance and Predictive Uncertainty

As is explained elsewhere in the present chapter, PREDVAR-suite utilities calculate the error variance of a particular prediction (which may also be a parameter), whereas PREDUNC-suite utilities calculate the uncertainty of that prediction (or parameter). As is explained by Doherty (2015), uncertainty is an intrinsic quality of parameters and of the calibration dataset, and is obtained by conditioning a pre-calibration (i.e. prior) parameter uncertainty covariance 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 using truncated singular value decomposition as an inverse problem solution device; truncation takes place at that singular value at which the error variance of the prediction of interest is minimized. It can be shown that posterior predictive (or parameter) uncertainty is always less than posterior predictive (or parameter) error variance; see section 7.4.1.3 of Doherty (2015).

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 for most models. 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.

10.17.4 The Prior Covariance Matrix

As is discussed by Doherty (2015), highly-parameterized error/uncertainty analysis requires that the user provide a covariance matrix of innate parameter variability; this is referred to as the C(**k**) matrix in Doherty (2015) and in equations provided in this manual. 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(**k**) 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(**k**) matrix is supplied to GENLINPRED through a parameter uncertainty file. The format of this file is discussed in section 2.5 of 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 from the PARCOV utility and from the PPCOV family of utilities supplied with the PEST Groundwater Data Utilities 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 that parameter.

10.17.5 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 notional calibration implemented using truncated singular value decomposition by the PREDVAR suite of programs; at the same time, they provide the basis for conditioning of pre-calibration parameter uncertainties 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 user-supplied PEST control file. For a linear model, this function is represented by its sensitivities to all adjustable parameters.

These sensitivities may be provided through a "sensitivity file". This file must adopt PEST matrix file format – see section 2.4 of this manual; sensitivities must be provided as a vector, that is, as a matrix with one column and with one row for each adjustable parameter. Alternatively predictive sensitivities may comprise one row of a Jacobian matrix, either the Jacobian matrix that complements the PEST control file which defines the inverse problem, or another Jacobian matrix altogether that was produced on the basis of a PEST control file that was built specifically for obtaining the sensitivities of one or a number of predictions to the same adjustable parameters as those that are employed in the inversion 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 inversion 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.

10.17.6 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 GENLINPRED issues many prompts, and it is easy to make a mistake in responding to one of them. 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 you with two options for prompt sequences. As will be seen shortly, through one of its early prompts GENLINPRED provides you with the option of receiving an abbreviated set of prompts. If this option is selected, GENLINPRED does not ask whether you wish to undertake parameter estimability analysis; instead it assumes 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 means of enhancing ease of use, a default response is provided for most of GENLINPRED's prompts; by pressing the <Enter> key, the default is accepted. You 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 measures of pre-calibration parameter uncertainty.

As a final means to expedite GENLINPRED user interaction, GENLINPRED writes an optional "response file". This records your responses to each of its prompts. Suppose that this file is named *genlinpred.rsp* (it is your choice). Then after it has been run on the basis of

keyboard input supplied by you in response to each of its individual prompts, GENLINPRED 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
```

Figure 10.8 Example of a GENLINPRED response file.

10.17.7 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.

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 GENLINPRED 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 was mentioned 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 for a C(**k**) 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 you respond with "b", GENLINPRED builds a C(**k**) 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 your 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 it 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 your 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 the inverse of each of them to thereby equal the uncertainty associated with the measurement to which it is assigned. The square of 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.

The easiest way to ensure that weights are at least approximately equal to measurement/structural uncertainty (so that a response of "y" can be provided for the above prompt) is to use the PWTADJ2 utility. This builds a PEST control file in which weights are calculated to be commensurate with misfit attained through a previous calibration exercise.

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" you are 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, you have the ability to over-ride SUPCALC's calculations (see below). Alternatively, you may supply the dimensions of the calibration solution space yourself 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 and/or observations 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 "n", 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 you wish 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 to all adjustable parameters featured in the inverse problem 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, with sensitivities recorded as a vector, i.e. as a matrix with a single column.

You are 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 latter is recommended, for the former 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 GENLINPRED's analysis. Note also that calculation of contributions to uncertainty is a far less numerically intensive procedure than calculation of contributions to error as the latter requires calculation of error variance at many different singular values so that it can be minimized; also uncertainty is a more "natural" concept than error.

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 to compute worth of groups of observations or individual observations is selected 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 PREDVAR5 and PREDUNC5; GENLINPRED uses both of these. 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. Note that calculation of worth with respect to uncertainty

is a far less numerically intensive procedure than calculation of worth with respect to error; also uncertainty is a more "natural" concept than error.

GENLINPRED then gets down to work. Regardless of your 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 whether you 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 your 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-suite programs) no formal subdivision of parameter space into solution and null spaces is required. In contrast, where error variance analysis is undertaken (by members of the PREDVAR suite), the predictive error variance is minimised with respect to singular value number on each occasion that predictive error variance is calculated; this can be a numerically intensive procedure.

10.17.8 Error and Uncertainty Tables

An inspection of tables produced by GENLINPRED may reveal the following.

1. Predictive error variance is greater than predictive uncertainty variance. (This is in harmony with theory; see section 7.4.1.3 of Doherty, 2015.)
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 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 singular value decomposition calibration exercise through which it is assessed. Furthermore, it is not a Bayesian procedure; error limits are less statistically efficient than are uncertainty limits. Nevertheless, it imitates what happens in practice where a model is normally calibrated first and then used as a basis for predictive error analysis in lieu of predictive uncertainty analysis because implementation of Bayesian analysis in conjunction with highly-parameterized nonlinear models is a task that can attract an overwhelming numerical burden. In the linear context, however, things are different, for in this context parameter and predictive uncertainties can be calculated relatively easily (provided the Gaussian assumption is valid). Hence the outcomes of uncertainty analysis should be used in preference to the outcomes of error analysis when assessing parameter importance and observation worth.

10.17.9 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 approximate, with a tendency for it to err on the side of too large 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. This diminishes the information content of the calibration dataset, or at least the information that can be transferred to model parameters. This, in turn, expands the null space to a higher dimensionality than that calculated by SUPCALC under the assumption of limited measurement noise correlation (an assumption that is implied in the use of weights for observations rather than a covariance matrix). Hence there may be occasions when you should 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).

10.17.10 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 made 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 contributions that different parameter types make to the errors/uncertainties of critical predictions required of a model; and
2. the parameter types that still contribute significantly to these errors/uncertainties, 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) becomes correlated through the parameter estimation process. 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. 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 it only influences this prediction indirectly through calibration-induced parameter correlation.

10.17.11 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-suite programs, and hence with GENLINPRED, may be high. Furthermore, there may be some occasions where use of one of these equations (particularly the low observation number option) can induce numerical errors due to the large number of matrix operations involved. Contact me if GENLINPRED computes obviously erroneous quantities such as negative uncertainties and negative parameter contributions to predictive uncertainty and I will alter GENLINPRED's internal settings to alter its choice of equation.

10.17.12 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, or should numerical error arise because of inappropriate choice of the PREDUNC uncertainty equation, you should run the programs that GENLINPRED runs yourself (that is, 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 utility suites. (Note that there is nothing to be gained by running SCALEPAR prior to running a PREDUNC-suite program.)

10.18 GENLINPRED_ABBREV

GENLINPRED_ABBREV is a version of GENLINPRED that is easier to use than GENLINPRED because it asks fewer less questions. Differences with the full version of GENLINPRED are as follows:

- GENLINPRED_ABBREV does not calculate relative parameter error variance reduction. It only calculates relative parameter uncertainty variance reduction.
- GENLINPRED_ABBREV does not ask whether you would like to calculate parameter contributions to predictive error, or data worth in terms of predictive error. These operations are not performed. However it does calculate parameter contributions to predictive uncertainty, and it does calculate data worth in terms of predictive uncertainty.
- GENLINPRED_ABBREV does not give you the option of calculating contributions to predictive uncertainty by individual parameters. It calculates contributions to predictive uncertainty only for parameter groups (without asking whether you would like this or not).
- GENLINPRED_ABBREV does not give you the option of assessing data worth for individual observations. It calculates data worth only for observation groups (without asking whether you would like this or not).
- GENLINPRED_ABBREV does not give you the opportunity to over-ride SUPCALC's estimation of the dimensionality of the solution space, unless SUPCALC specifies that this is zero.
- GENLINPRED_ABBREV does not write a response file which can be used for later GENLINPRED_ABBREV runs.

In all other respects, use of GENLINPRED_ABBREV is the same as that of GENLINPRED.

11. Linear Error and Uncertainty – Part II

11.1 Introduction

11.1.1 General

The utility programs that are documented in this chapter are older than those documented in the previous chapter. Many have not been used for a while, and they have not been upgraded over time as other utilities, and PEST itself, have been updated. Nevertheless they all still work, and some users may find them helpful. They focus on parameter and predictive error rather than uncertainty. They rely on information forthcoming from a PEST run in which a model was actually calibrated. Their analyses are therefore somewhat less general in nature than those presented in the previous chapter.

11.1.2 Concepts

Post-Calibration Parameter and Predictive Error

Let **k** represent parameters employed by a model. Let $\underline{\mathbf{k}}$ represent the parameter set achieved through model calibration. From equation 5.5.10 of Doherty (2015) the covariance matrix of post-calibration parameter error is calculated as

$$\mathrm{C}(\underline{\mathbf{k}} - \mathbf{k}) = (\mathbf{I} - \mathbf{R})\mathrm{C}(\mathbf{k})(\mathbf{I} - \mathbf{R})^t + \mathbf{G}\mathrm{C}(\boldsymbol{\varepsilon})\mathbf{G}^t \qquad (11.1.1)$$

where

- **k** represents "true" model parameters (which are unknown);
- $\underline{\mathbf{k}}$ represents calibrated model parameters;
- C(**k**) represents the prior parameter covariance matrix, this describing the innate variability of real-world parameters;
- C(**ε**) represents the covariance matrix of measurement/structural noise, mostly assumed to be diagonal;
- **R** is the so-called "resolution matrix"; and
- **G** is the matrix through which estimated parameter values (i.e. the elements of $\underline{\mathbf{k}}$) are calculated from measurements (which are denoted as **h** in Doherty, 2015); **G** is referred to herein as the "parameter solution matrix", or more simply as the "**G** matrix".

R is related to **G** through the equation

$$\mathbf{R} = \mathbf{G}\mathbf{Z} \qquad (11.1.2)$$

where **Z** specifies the relationship between model parameters and model outputs used in the calibration process (see equation 10.3.6).

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

$$s = \mathbf{y}^t\mathbf{k} \qquad (11.1.3a)$$

while its counterpart as calculated by the calibrated model is

$$\underline{s} = \mathbf{y}^t\underline{\mathbf{k}} \qquad (11.1.3b)$$

Model predictive error is then obtained as

$$\underline{s} - s = \mathbf{y}^t(\underline{\mathbf{k}} - \mathbf{k}) = -\mathbf{y}^t(\mathbf{I} - \mathbf{R})\mathbf{k} + \mathbf{y}^t\mathbf{G}\boldsymbol{\varepsilon} \qquad (11.1.4)$$

while model predictive error variance (i.e. the "variance of potential wrongness" of a prediction made by a calibrated model) is given by

$$\sigma^2_{\underline{s}-s} = \mathbf{y}^t(\mathbf{I} - \mathbf{R})\mathrm{C}(\mathbf{k})(\mathbf{I} - \mathbf{R})^t\mathbf{y} + \mathbf{y}^t\mathbf{G}\mathrm{C}(\boldsymbol{\varepsilon})\mathbf{G}^t\mathbf{y} \qquad (11.1.5)$$

Note that while derivation of these equations rests on an assumption of model linearity, they are nevertheless approximately correct when applied to many nonlinear models. With some modifications, they can also be used as a basis for nonlinear analysis; PEST's null space Monte Carlo methodology rests on equation 11.1.1.

*Calculation of the **R** and **G** Matrices*

Formulas for **R** and **G** featured in the above equations depend on the method used by PEST to solve the inverse problem. For an overdetermined system, for which the regularisation opportunities offered by truncated singular value decomposition and Tikhonov schemes are not required, the resolution matrix **R** is simply **I**, the identity matrix. However in many real world cases, barely overdetermined problems are rescued from numerical instability through PEST's use of a high-valued Marquardt lambda (which is a de-facto Tikhonov regularisation device). Even where mathematical regularisation is formally introduced to a real world problem, it is often insufficient to guarantee unequivocal numerical stability of solution of that problem; hence PEST will often respond by raising the value of the Marquardt lambda, this allow progress in solution of an inverse problem to be made when it may otherwise founder.

The role of the Marquardt lambda in calculation of the **R** and **G** matrices is recognised in the utility software described in the present chapter. It must be noted, however, that the Marquardt lambda is not a very good regularisation device; in many cases it can lead to inverse problem solutions which are not of minimised error variance, at the same time as it can lead to distortion of the resolution matrix. Hence, whether using one of the specialist regularisation devices offered by PEST to assist in solution of an inverse problem, or whether using manual regularisation to promulgate inverse-problem well-posedness, attempts should be made to keep the Marquardt lambda low.

Formulas used for calculation of **R** and **G** are now provided. Variables appearing in these formulas are as follows.

Z This is the Jacobian matrix, which is a linearization of the relationship between model parameters and model outputs used in the calibration process. Each row of the Jacobian matrix provides derivatives of a particular model outcome for which there is a complementary field measurement with respect to all adjustable parameters. If a parameter is log-transformed, pertinent elements of **Z** pertain to the log of that parameter. Note that for SVD-assisted inversion the **Z** matrix refers to base parameters, not super parameters in the equations below.

X This is the super-parameter Jacobian matrix whose elements are derivatives of model outcomes with respect to super parameters estimated through SVD-assisted inversion.

λ The PEST-calculated Marquardt lambda.

T The matrix of Tikhonov regularisation constraints. These constraints are assumed to be of the form $\mathbf{Tk} = \mathbf{0}$.

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 which constitute the calibration dataset. $\boldsymbol{\varepsilon}$ featured in equations 11.1.1, 11.1.4 and 11.1.5 is the "noise" or "measurement error" associated with these observations.

Q The observation weight matrix (calculated from user-supplied weights and measurement covariance matrices).

V The matrix whose columns are orthogonal unit eigenvectors of $\mathbf{Z}^t\mathbf{QZ}$ (and hence of $\mathbf{Q}^{1/2}\mathbf{Z}$) as calculated through singular value decomposition undertaken either during every iteration of the parameter estimation process (when this is achieved through truncated singular value decomposition), or at the beginning of the inversion process for determination of super parameters (if using SVD-assisted parameter estimation).

E A diagonal matrix whose elements are the eigenvalues of $\mathbf{Z}^t\mathbf{QZ}$ (arranged in decreasing order) determined through singular value decomposition. The singular values of $\mathbf{Q}^{1/2}\mathbf{Z}$ are the square roots of these.

$\mathbf{V}_1$ 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 $\mathbf{Z}^t\mathbf{QZ}$.

The utility software described below through which **R** and **G** can be calculated employs the **Z** (or **X** in the case of SVD-assisted inversion) matrix corresponding to the best parameter set achieved through the parameter estimation process. This is stored in file *case.jco* where *case* is the filename base of the PEST control file. Like *case.rsd* (see below) and *case.par* (the parameter value file), *case.jco* is updated by PEST whenever an improved parameter set is obtained. The **Z** matrix used by SVD-assist, 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 for different regularisation methodologies used to solve the inverse problem are now presented. (Note that the use of LSQR in solution of the inverse problem is not accommodated by the utility programs discussed in this chapter.)

Overdetermined Parameter Estimation (with no Regularisation)

$$\mathbf{R} = (\mathbf{Z}^t\mathbf{QZ} + \lambda\mathbf{I})^{-1}\mathbf{Z}^t\mathbf{QZ} \qquad (11.1.6a)$$

$$\mathbf{G} = (\mathbf{Z}^t\mathbf{QZ} + \lambda\mathbf{I})^{-1}\mathbf{Z}^t\mathbf{Q} \qquad (11.1.6b)$$

Tikhonov Regularisation

$$\mathbf{R} = (\mathbf{Z}^t\mathbf{Q}\mathbf{Z} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{Z}^t\mathbf{Q}\mathbf{Z} \quad (11.1.7a)$$

$$\mathbf{G} = (\mathbf{Z}^t\mathbf{Q}\mathbf{Z} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{Z}^t\mathbf{Q} \quad (11.1.7b)$$

Singular Value Decomposition with Zero Marquardt Lambda

$$\mathbf{R} = \mathbf{V}_1\mathbf{V}_1^t \quad (11.1.8a)$$

$$\mathbf{G} = \mathbf{V}_1\mathbf{E}_1^{-1}\mathbf{V}_1^t\mathbf{Z}^t\mathbf{Q} \quad (11.1.8b)$$

SVD-Assist

$$\mathbf{R} = \mathbf{V}_1(\mathbf{X}^t\mathbf{Q}\mathbf{X} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q}\mathbf{Z} \quad (11.1.9a)$$

$$\mathbf{G} = \mathbf{V}_1(\mathbf{X}^t\mathbf{Q}\mathbf{X} + \beta^2\mathbf{T}^t\mathbf{S}\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{Q} \quad (11.1.9b)$$

The following should be noted.

1. PEST allows various combinations of different regularisation schemes to be used in estimating 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 in this chapter.
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 inversion process.

11.1.3 Some Special Considerations

Regularisation Relationships

As mentioned above, where Tikhonov regularisation is employed it is assumed to be of the type

$$\mathbf{T}\mathbf{k} = \mathbf{0} \quad (11.1.10)$$

In PEST, regularisation can be linear (supplied through prior information equations), or nonlinear (supplied as observations). In both cases these are identified as regularisation relationships through being assigned to an observation group whose name begins with "regul" or is identified as a regularisation group using the "regulgroup" string in the "observation groups" section of the PEST control file. However PEST also allows regularisation relationships of the following type to be supplied

$$\mathbf{T}\mathbf{k} = \mathbf{j} \quad (11.1.11)$$

Calculation of the resolution matrix, as implemented in the utility software described below, cannot accommodate relationships of the type expressed by equation 11.1.11. In most cases, equation 11.1.11 can be transformed to equation 11.1.10 by appropriate parameter re-definition.

Initial Parameter Values

When using truncated singular value decomposition or SVD-assisted inversion, the integrity of the predictive error variance analysis process requires that initial parameter estimates (provided in the "parameter data" section of the PEST control file) correspond to most likely parameters according to a user's concept of parameter likelihood based on the current modelling context and the characteristics of the modelled area. That is, they constitute "minimum error variance" estimates of parameter values based on expert knowledge alone.

*The **Z** Matrix in SVD-Assisted Inversion*

As mentioned above, the **Z** matrix appearing in equation 11.1.9a provides the sensitivities of model outputs for which there are corresponding field measurements to base parameters. In SVD-assisted inversion these can far outnumber super parameters; computation of the **Z** matrix can therefore be costly. Nevertheless, as described in part I of this manual, this matrix must be calculated (based on initial parameter values) prior to undertaking SVD-assisted inversion, 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 equation 11.1.9a 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 or -2 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 base parameter sensitivities, calculated using optimised parameter values.

11.1.4 PEST Requirements

The IRES Variable

As part of its normal suite of output files, PEST writes a "resolution data file" named *case.rsd* where *case* is the filename base of the PEST control file. If desired, writing of this file can be enabled or suppressed using the IRES variable featured on the last line of the "control data" section of the PEST control file. If IRES is omitted, its value is assumed to be one if PEST is run in "regularisation" mode and/or if PEST's singular value decomposition 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 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 11.1.6 to 11.1.9.

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 mistaken 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.

11.1.5 Program Versions

At the time of writing only 32 bit versions of the utility programs documented in the present chapter are available. Hence they may not work for inversion problems which involve high numbers of parameters and high numbers of observations. Contact me for a 64 bit version of any of these utilities if you need them.

11.2 RESPROC

11.2.1 General

RESPROC stands for "resolution data postprocessor". It is designed to be used after completion of a PEST run. Normally PEST will have been run in "regularisation" mode, or with SVDMODE set to 1, will have undertaken SVD-assisted inversion, or will have implemented all of these aspects of its inversion functionality. However RESPROC can also provide useful results after PEST has been run in "estimation" mode without the use 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. These can then be used by utility programs documented below for calculation of parameter and predictive error variances. In order to save disk space, RESPROC writes the **R** and **G** matrices to binary files; however these matrices can be rewritten in ASCII form if desired using the RESWRIT utility described below.

11.2.2 What RESPROC Does

RESPROC reads the following files, all associated with an existing PEST dataset characterised by the *case* filename base:

1. a PEST control file (named *case.pst*);
2. a resolution data file (named *case.rsd*);
3. a 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 up to 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* for present purposes);
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 inversion 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.

11.2.3 Running RESPROC

RESPROC is run using the command

```
resproc case outfile [/(n)L]
```

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. (This file is referred to herein as a "binary resolution matrix file".)

Optionally a switch can accompany command line arguments. This can be supplied "/L" or "/nL". If the "/L" switch is used, then formulas 11.1.6 to 11.1.9 are used in calculation of **R** and **G** as shown. If the "/nL" switch is employed then the Marquardt lambda is set to zero for the purpose of calculating **R** and **G**. If this switch is omitted, "/L" is assumed.

As it executes, RESPROC writes its current activities to the screen. As discussed above, these activities involve manipulation of possibly large matrices. They also involve matrix inversion and possibly singular value decomposition (which will need to be undertaken twice if the previous inversion process 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 these matrices in its output binary resolution matrix file, records this fact to the screen, and ceases execution.

11.2.4 SVD-Assisted Inversion

Where the previous PEST run implemented SVD-assisted inversion, 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 specify the name of the PEST control file for this case as *bcase.pst*.
2. Base parameter sensitivities are calculated for this base case. These are 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 estimated parameter values are employed as initial parameter values. The command is

   ```
   parrep bcase.bpa bcase.pst bcase1.pst
   ```
6. NOPTMAX is set to -1 or -2 in *bcase1.pst*. Thus it calculates the Jacobian matrix (the **Z** matrix of equation 11.1.9a), and then ceases 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 suggests that use of a **Z** matrix calculated on the basis of optimised 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.

11.3 RESWRIT

The **R** and **G** matrices recorded by the RESPROC utility are not readable by the user. If it is desired that these matrices be subject to inspection and/or plotting, they can be converted to ASCII format using the RESWRIT utility.

RESWRIT is run using the command

```
reswrit resprocfile matfile1 matfile2
```

where

resprocfile	is the name of a binary 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.

The matrix files written by RESPROC respect the formatting specifications outlined in section 2.4 of this manual.

11.4 PARAMERR

11.4.1 General

PARAMERR constructs the parameter error covariance matrix C(**k**-**k**) of equation 11.1.1. It stores the two terms on the right side of equation 11.1.1 in separate files. This saves you from having to re-compute both of these terms if an input required by only one of them (for example C(**k**) or C(**ε**)) is altered. It also allows you to quantify the individual contribution to overall predictive error variance made by "uncaptured system heterogeneity" on the one hand (the first term), and measurement/structural error on the other hand (the second term). The first diminishes as the level of fit between model outputs and field measurements increases, while the second term grows as a higher level of model-to-measurement fits is attained; see Doherty (2015) for further details.

11.4.2 Using PARAMERR

PARAMERR receives information from keyboard input supplied by the user in response to a series of prompts. (It requires more information than can be easily 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 binary file generated by RESPROC which holds the **R** and **G** matrices should be provided. Next PARAMERR asks

```
Options are as follows:-
  to compute (I-R)C(k)(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 11.1.1 requires computation. However if both are required, enter 3. If this is done, PARAMERR's next prompts is

```
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:
```

Alternatively, if your choice is 1, then only the first two prompts are issued while if your response is 2 then only the third and fourth prompts are issued.

The parameter uncertainty file requested by the first of the above four prompts contains the prior parameter covariance matrix C(**k**). The observation uncertainty file requested by the third of the above four prompts contains the covariance matrix of measurement/structural noise C(**ε**). These must be prepared by you in accordance with specifications set out in section 2.5 of this manual. The covariance matrix output file whose name is supplied in response to the second of the above four prompts comprises the first term on the right side of equation 11.1.1, that is $(\mathbf{I} - \mathbf{R})\mathrm{C}(\mathbf{k})(\mathbf{I} - \mathbf{R})^t$. The covariance matrix output file whose name is supplied in response to the fourth of the above four prompts comprises the second term on the right of equation 11.1.1, that is $\mathbf{G}\mathrm{C}(\boldsymbol{\varepsilon})\mathbf{G}^t$. These are written in PEST matrix file format following the protocol discussed in section 2.4 of this manual. Each of these matrices is square with dimensions $m \times m$ where m is the number of adjustable parameters pertaining to the current parameter estimation problem.

11.5 PREDERR

11.5.1 General

PREDERR is similar in many respects to PARAMERR. Like PARAMERR it reads a parameter uncertainty file and an observation uncertainty file; see section 2.5 of this manual for specifications of these files. However rather than calculating and storing the two components of C(**k**–**k**), it calculates the error variance of a user-specified prediction. Calculation of C(**k**–**k**) is bypassed, this enhancing the efficiency of calculation of predictive error variance considerably. Hence unless calculation of C(**k**-**k**) is specifically required for a certain application, use of PREDERR is preferred over use of PARAMERR followed by matrix/vector manipulation utilities such as JROW2VEC and MATQUAD for calculation of the error variance of a certain prediction.

(Actually, as stated at the beginning of this chapter, the utility programs documented in the previous chapter, rather than those documented in the present chapter, are the preferred options for computing predictive error variance, as well as predictive uncertainty.)

11.5.2 Using PREDERR

Many of PREDERR's prompts are similar to those of PARAMERR. However unlike PARAMERR, 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 actually treated as a prediction for the purpose of error variance analysis undertaken by PREDERR using equation 11.1.5. 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. As is discussed extensively in this manual, the Jacobian matrix file will have been written by PEST. It is a binary 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 matrix 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 11.1.5 (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

 ************************************************************
```

Figure 11.1 Screen output of PREDERR.

11.6 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 (i.e. the **R** matrix) on the one hand and the solution matrix (i.e. the **G** matrix) on the other. These files must be in matrix file format as described in section 2.4 of this manual. If they were written by the PARAMERR utility then they will, indeed, adhere to this format.

Note the following.

1. Row names provide in the **R** 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 inverse problem.
3. Column names cited in the **G** matrix must pertain to observations employed in the current inverse problem.

11.7 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 multiple predictions featured in this file. Like PREDERR1 it prompts for a resolution matrix file, a solution 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". This is the file that it writes. An example appears in figure 11.2.

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

Figure 11.2 Part of a PREDERR2 output file.

The "variance_1" and "variance_2" terms appearing in the column headers of figure 11.2 are the **I-R** and **G** terms of the predictive error variance equation, i.e. equation 11.1.5. Thus these

terms describe the contributions to predictive error variance arising from "uncaptured heterogeneity" and measurement noise respectively. The contents of the "total" column are the sum of these two terms; the square roots of these totals comprise the "standard_dev" column.

11.8 PREDERR3

PREDERR3 is almost identical to PREDERR2. However instead of reading the **R** and **G** matrices from separate files, it reads both of these from an unformatted RESPROC output file.

11.9 REGERR

11.9.1 General

REGERR evaluates the covariance matrix of regularisation-induced model output error. For present purposes this is defined as

$$\boldsymbol{\tau} = \mathbf{Z}(\mathbf{I} - \mathbf{R})\mathbf{k} \qquad (11.9.1)$$

so that

$$C(\boldsymbol{\tau}) = \mathbf{Z}(\mathbf{I} - \mathbf{R})C(\mathbf{k})(\mathbf{I} - \mathbf{R})^t\mathbf{Z}^t \qquad (11.9.2)$$

In this equation $C(\mathbf{k})$ is, as usual, the pre-calibration parameter covariance matrix, **R** is the resolution matrix for the current inverse problem, and **Z** is the Jacobian matrix. REGERR obtains **R** from a RESPROC output file and **Z** from a Jacobian matrix file (i.e. a JCO file). Presumably these will both pertain to 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.

11.9.2 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: pestcase.rpo
Enter name of parameter uncertainty file: param.unc
Enter name of Jacobian matrix file: pestcase.jco
Enter name for output covariance matrix file: cov.mat
- reading RESPROC output file pestcase.rpo...
- file pestcase.rpo read ok.

- reading Jacobian matrix file pestcase.jco...
- Jacobian matrix file pestcase.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 in section 2.5 of this manual. The covariance matrix file written by REGERR employs the matrix file protocol discussed in section 2.4 of this manual.

12. Nonlinear Error and Uncertainty

12.1 Introduction

Programs described in this chapter assist in the implementation of nonlinear error and uncertainty analysis. Some of them explore uncertainty (or error variance as a surrogate) through generation of random parameter sets. This is facilitated using the RANDPAR utility. By first sampling a prior covariance matrix, and by then running the model many times using these samples, the prior uncertainty of a prediction can be established. Alternatively, posterior uncertainty can be explored by sampling a linear approximation to the posterior parameter covariance matrix (produced, for example, using the PREDUNC7 utility described in a previous chapter). Another option is to modify samples of the prior parameter distribution in order to ensure respect for calibration constraints using subspace methods; see the PNULPAR utility. Thus-obtained samples of a linear approximation to the posterior parameter distribution can then be adjusted to adhere more strongly to calibration constraints using PEST's null space Monte Carlo methodology described in section 10.6.3 of part I of this manual.

If a prediction is made by a model using many different parameter sets, the values computed for the prediction can be processed in order to formulate a prediction value histogram. This histogram can be considered to be an approximation to the predictive probability density function. The RDMULRES, MULPARTAB and COMFILNME utilities can assist in this type of data collation.

Calibration-constrained direct predictive maximisation/minimisation provides an alternative to Monte Carlo exploration of post-calibration predictive uncertainty. This is implemented by PEST's predictive analyser; see part I of this manual. Where parameter numbers are few, and where an inverse problem is well-posed, PEST's predictive analyser can implement this procedure in a relatively efficient manner. Unfortunately, however, this methodology becomes less efficient in highly-parameterized contexts. Nevertheless assistance in file setup for implementation of this procedure in the highly parameterised context can be gained through use of the REGPRED utility.

As is also described in part I of this manual, direct hypothesis testing of predictive possibilities can be undertaken by running PEST in "pareto" mode. A prediction is deemed to be unlikely when it is demonstrably incompatible with either or both of expert knowledge and the historical behaviour of the system under investigation. The compatibility (or otherwise) of a parameter set with expert knowledge (as encapsulated in a covariance matrix) can be tested using the ASSESPAR utility.

Utility programs described in the present chapter are complemented to some extent by those discussed in the following chapter; the latter provide the means to undertake Latin hypercube sampling of a parameter probability density function. Some of the utilities described in the present chapter can be used in conjunction with those described in the next, particularly for postprocessing of model runs that are collectively undertaken on the basis of Latin hypercube samples of a pre- or post-calibration parameter probability density function.

12.2 RANDPAR

12.2.1 General

RANDPAR generates parameter value realisations using a random number generator. The parameters for which it must generate realisations are read from an existing PEST control file. RANDPAR-generated random numbers are inserted into a series of PEST parameter value files (see section 2.2 of this manual for specifications of this type of file) 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 0 in each of these files, the objective function corresponding to each set of parameter values can be calculated.

You can choose between two probability distributions for random number generation. The first option is 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 section 2.5 of this manual for specifications 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 some parameters are 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; values supplied by RANDPAR for these parameters are the same as in the PEST control file. Nor are random values generated for tied parameters; instead, the values assigned to tied parameters are such that the ratios of initial values between tied parameters and those to which they are linked as supplied in the original PEST control file are 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.

12.2.2 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. As stated above, 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.

Suppose that you respond to the above prompt with "u". Then 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 defined 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]:
```

As is apparent from the above prompt, there are 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 its mean value. 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). An "e" in response to this prompt selects the former option while an "m" selects the latter option.

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 its lower bound or above its upper bound as recorded in the PEST control file, RANDPAR will assign the parameter a value equal its lower or upper bound respectively. Alternatively, if the answer is "n", bounds are 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 is 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 section 2.5 of this manual. 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
```

Figure 12.1 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. Suppose that one or more of the ordering relationships expressed in a parameter ordering file are not respected by a RANDPAR-generated random parameter set. Then RANDPAR will abandon that realisation, and will continue to generate further 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 parameter 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* ….*baseN.par*, where *N* is the number supplied in response to the second of the above prompts.

12.3 RANDPAR1

Use of RANDPAR1 is identical to that of RANDPAR. It uses a different methodology for generation of random correlated parameter fields which may be more efficient where parameter numbers are large.

12.4 RANDPAR2

RANDPAR2 has similar functionality to that of RANDPAR1. However random parameter realizations are recorded in a comma-delimited file (i.e. a CSV file). This file can provide prior parameter realizations to the PESTPP-IES program from the PEST++ suite.

Typical RANDPAR2 prompts and responses are as follows:

```
Enter name of existing PEST control file: case.pst
- 402 parameters read from file case.pst.
- 402 of these are adjustable.

Use (log)normal or (log)uniform distrib for param generation? [n/u]: n
Compute means as existing param values or range midpoints? [e/m]: e
Respect parameter ranges? [y/n]: y
```

```
Enter name of parameter uncertainty file: param.unc
- parameter uncertainty file param.unc read ok.

Enter name of parameter ordering file (<Enter> if none): <Enter>

Enter name for CSV file: random.csv
How many realizations do you wish to generate? 20
Add initial parameters as base realization? [y/n]: y
Realizations are rows or columns in CSV file? [r/c]: r

Enter integer random number seed (<Enter> if default): <Enter>

- undertaking singular value decomposition of covariance matrix...
- file random.csv written ok.
```

It is apparent from the above prompts that RANDPAR2 asks the user some questions that are not asked by RANDPAR1. One of these questions pertains to inclusion of a "base" parameter realization in the CSV file that it writes. This parameter set is not actually a parameter realization at all; it is actually the set of parameters that are featured as initial values in the PEST control file.

Each parameter realization is provided with a name. The name given to the initial parameter "realization" is "base", in accordance with the protocol used by PESTPP-IES. Also in accordance with the PESTPP-IES protocol, other realizations are given integer labels; integers are sequential and start at zero. If the base realization is included, then the number of random realizations appearing in the CSV file is reduced by 1, so that the total number of realizations is equal to that which the user asked for.

The CSV file written by RANDPAR2 can be oriented in one of two ways. The user selects that which is best for him/her through the way in which he/she answers the second-to-last of the questions asked by RANDPAR2. The values of all parameters that comprise a single realization can be written along a row; other realizations then occupy other rows. Alternatively, the values of all parameters that comprise a realization can occupy a column; other realizations then occupy other columns. Where parameter numbers are high, the first option may result in CSV files which are difficult to import into a spreadsheet.

12.5 RANDPAR3

Like other members of the RANDPAR* family, RANDPAR3 generates random realizations of parameters. In fact, its behaviour is very similar to that of RANDPAR2. However instead of recording random parameter realizations in a CSV file, RANDPAR3 records them in a JCB file (i.e. an "enhanced Jacobian matrix file" of the type used by programs of the PEST++ suite). This file is readable by PESTPP-IES; hence parameter fields generated by RANDPAR3 can be used by this program as initial parameter ensembles.

In a similar manner to RANDPAR2, RANDPAR3 prompts:

```
Add initial parameters as base realization? [y/n]:
```

As was discussed in documentation of RANDPAR2, the parameter "realization" which is comprised of initial parameter values recorded in the PEST control file is given the name "base". Unlike RANDPAR2, however, RANDPAR3 does not ask the user for the roles of rows and columns in the file which it writes. Each row of the JCB file written by RANDPAR3 houses a single parameter realization, this being in accordance with PESTPP-IES's expectations of this type of file.

12.6 RANDPAR4

12.6.1 General

RANDPAR4 is modified from RANDPAR3. It fills JCB files with parameter increment ensembles. These are used by the randomized Jacobian functionality of PEST_HP. For parameters that are log-transformed, RANDPAR4 generates random parameter multipliers. For parameters that are not log-transformed, it generates random additive increments. Parameters which are tied or fixed in the PEST control file which it reads are not provided with increments at all, and hence do not feature in a RANDPAR4-generated JCB file.

The statistical properties of random parameter increments are read from a parameter uncertainty file; see section 2.5 of this manual for a description of this file type. The statistical properties awarded to parameter increments in this file should pertain to their logs (to base 10) for parameters that are log transformed, and to their native values for parameters that are not log transformed. (This is the usual protocol for a parameter uncertainty file, regardless of whether it is being used as a basis for random parameter value generation, or for generation of random parameter increments.) Using these stochastic properties, RANDPAR4 generates random realizations of parameter increments centered on 0.0. Then, for log-transformed parameters, these increments are raised to the power 10 in order to become multipliers centered on 1.0.

12.6.2 Using RANDPAR4

Upon commencement of execution, RANDPAR4 asks:

```
Enter name of existing PEST control file:
```

After perusing this file, RANDPAR4 next asks for the name of the parameter uncertainty file that it must read in order to ascertain the stochastic properties of parameter increments. (Note that there is no reason why this file cannot also express the prior uncertainties of parameters themselves; the size of the increments which RANDPAR4 generates can be reduced by pertinent settings provided to PEST_HP when this file is actually read by PEST_HP.) The prompt is:

```
Enter name of parameter uncertainty file:
```

Next RANDPAR4 prompts for the name of the JCB file that is must write, and for the number of parameter increment realizations that it must record in this file:

```
Enter name for JCB file:
How many realizations do you wish to generate?
```

Finally, RANDPAR4 prompts for a seed for its random number generator:

```
Enter integer random number seed (<Enter> if default):
```

RANDPAR4 then generates the required ensemble of random parameter increments, and ceases execution.

12.7 PNULPAR

12.7.1 General

PNULPAR can be used in conjunction with RANDPAR to undertake calibration-constrained Monte-Carlo analysis. In general, RANDPAR-generated parameter fields do not respect

calibration constraints; PNULPAR can modify these parameter fields so that they do.

PNULPAR reads a set of parameter value files generated by RANDPAR. The sets of random parameters contained within these files may have been generated on the basis of a PEST control file in which initial parameter values are expected parameter values from an expert knowledge point of view. Alternatively, the PEST control file on which RANDPAR parameter set generation was based may have housed calibrated parameter values. (In the latter case the PARREP utility may have been used to replace pre-calibration initial parameter values with calibrated parameter values in the PEST control file.)

Regardless of which of the above alternatives was employed for RANDPAR-based random parameter set generation, use of PNULPAR requires that a PEST control file exists in which initial parameter values are calibrated parameter values. This file is referred to herein as a "post-calibration PEST control file"; it may or may not be the same PEST control file as that used by RANDPAR for random parameter set generation. A corresponding JCO file must exist for the post-calibration PEST control file. 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 created from another JCO file using the JCO2JCO utility.

PNULPAR reads the set of parameter value files produced by RANDPAR. It writes a new set of parameter value files in which parameter values are modified from those occurring in the RANDPAR-generated files. In generating this new set of parameter value files PNULPAR undertakes the following operations.

1. It undertakes singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ where $\mathbf{Q}$ is the observation weight matrix and $\mathbf{J}$ is the Jacobian matrix pertaining to the post-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 $\mathbf{V}$ matrix forthcoming from singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ whose corresponding singular values are significantly non-zero. Recall that singular value decomposition of $\mathbf{Q}^{1/2}\mathbf{J}$ is described by the following formula.

   $$\mathbf{Q}^{1/2}\mathbf{J} = \mathbf{U}\mathbf{S}\mathbf{V}^{t} \qquad (12.7.1)$$

   As described by Doherty (2015), the matrix $\mathbf{V}$ can be partitioned as $[\mathbf{V}_1\ \mathbf{V}_2]$ where $\mathbf{V}_1$ is an orthonormal matrix whose columns span the calibration solution space, and $\mathbf{V}_2$ is an orthonormal matrix whose columns span the calibration null space. The dimensionality of the calibration solution and null spaces can be estimated with the help of utilities such as SUPCALC.

2. Suppose that elements of the vector $\mathbf{k}$ comprise the set of parameter values contained within a RANDPAR-generated parameter value file. Let $\underline{\mathbf{k}}$ describe the set of parameters contained (as initial parameter values) within the post-calibration PEST control file read by PNULPAR; as stated above, it is assumed that the vector $\underline{\mathbf{k}}$ was calculated during a previous calibration process, and thus comprises the minimum error variance estimate of the real (but unknown) parameter set $\mathbf{k}$. PNULPAR next computes $\mathbf{k}$-$\underline{\mathbf{k}}$ taking parameter transformation status into account.

3. Components of parameter differences encapsulated in $\mathbf{k}$-$\underline{\mathbf{k}}$ that possess a non-zero projection onto the calibration solution space are then removed by computing a set of "projected parameter differences" $\mathbf{k}_d$ using the equation

   $$\mathbf{k}_d = \mathbf{V}_2\mathbf{V}_2^{t}(\mathbf{k}-\underline{\mathbf{k}}) \qquad (12.7.2)$$

4. A new set of parameter values is then produced by adding $\mathbf{k}_d$ to $\underline{\mathbf{k}}$. These values are

written to a new parameter value file.

With the exception of singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{J}$ 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.

PNULPAR preserves the ratios of tied parameters to parent parameters (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.)

PNULPAR handles differences in fixed parameter values slightly differently. If a fixed parameter has a different value in a RANDPAR-generated parameter value file from that which it has in the post-calibration PEST control file, the value in the parameter value file prevails. This strategy may be beneficial in certain instances of groundwater model calibration where, for example, the PPSAMP utility from the Groundwater Data Utility suite may be used instead of RANDPAR for generation of random parameter values. However differences in the values of fixed parameters should never occur if RANDPAR is used to generate random parameter values as it respects the values of fixed parameters, and does not generate random values for them.

12.7.2 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 parameter values calculated during a previous PEST run. (The PARREP utility may have been used to insert these into an existing PEST control file containing non-calibrated parameter values.) A JCO file corresponding to this post-calibration PEST control file must also exist. In many cases of PNULPAR usage, this same PEST control file will have formed the basis for RANDPAR random parameter set generation.

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. As stated above, SUPCALC can offer advice on this matter. Alternatively, if you would like to perform calculations on the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix yourself in order to determine an appropriate answer to the above question, PNULPAR can facilitate this process by storing the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix in PEST matrix file format. 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:
```

(Note that the "X" matrix referred to by PNULPAR is actually the **J** matrix used in present documentation.) Next PNULPAR asks

```
Enter filename base of existing parameter value files:
```

Suppose that your answer to the above question is "*basename*". Then PNULPAR will look for files *basename1.par, basename2.par* etc. When it encounters no more files in this sequence, it assumes that it has found them all. For each such existing 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 numerical 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", "regularisation" or "pareto". So if you wish to employ PNULPAR after undertaking regularised inversion (as is often the case), alter PESTMODE to "estimation" in the governing PEST control file and remove all regularisation observations and prior information equations from that file. (Alternatively, use the SUBREG1 utility to accomplish the same thing automatically.) 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; when undertaking singular value decomposition of the $\mathbf{Q}^{1/2}\mathbf{J}$ matrix to define calibration solution and null spaces, no regularisation information should be included in the **J** matrix.

12.7.3 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 similar to that achieved during the previous calibration exercise.

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 adjusted back into calibration with very little effort, this often requiring only one PEST iteration. (If desired, set NOPTMAX to 1 in the governing PEST control file to ensure that only one iteration actually takes place.) Furthermore, this process can be made even more inexpensive by employing sensitivities residing in an existing JCO file (for example, 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 999. 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 PHIRATSUF control variable very low (for example 0.001) and RLAMFAC to -4. Thus PEST will be forced to test a number of different Marquardt lambdas before the end of the iteration.

Hopefully, with the help of these strategies, an objective function can then be found which is as low as it can possibly be based on current sensitivities.

If the calibration null space is relatively large, then use of SVD-assist for calibration adjustment of each PNULPAR-generated parameter set can be achieved with a high level of model run efficiency, even if more than one iteration is required to reduce the objective function to a level at which the model is deemed to be "calibrated". See section 10.6.3 of part I of this manual for details.

Whichever of the above methods is chosen to enforce calibration constraints on PNULPAR-modified parameter sets, the higher that you inform PNULPAR is the dimensionality of the calibration solution space, the less work will be required to enforce these constraints. This is because the higher that PNULPAR believes the dimensions of the calibration solution space to be, the greater is the contribution made by the parameter field calculated during the previous calibration process to each PNULPAR-generated parameter field. The user-supplied solution space dimensionality therefore provides a "lever" through which you can increase the efficiency of attainment of parameter fields which respect calibration constraints (as embodied in respect for an objective function that is deemed to "calibrate" the model). The cost of such a speed-up strategy is, of course, some reduction in the diversity of the calibration-adjusted random parameter sets thus obtained.

12.8 RDMULRES

12.8.1 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.

Monte Carlo Analysis

Monte Carlo analysis of model predictive uncertainty can 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, for example a geostatistically-based parameter field generator such as the Stanford Geostatistical Modeling Software package SGEMS.

Suppose that RANDPAR (and possibly PNULPAR) was used to generate a suite of parameter value files named *parval1.par*, *parval2.par*, etc. Suppose that the model reads its parameters from a single input file 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 batch file shown in Figure 12.2 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 model.tpl model.in parval%%i.par
model.exe
copy model.out model%%i.out)
```

Figure 12.2 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. For each model run the model output file is linked to the parameter set on whose basis it was recorded by integer index (represented by the *%%i* variable). Note that other options could also be used for the commissioning of many model runs. For example the PARREP utility could be used to insert parameter values into a PEST control file. Then PEST could be run. If NOPTMAX is set to 0 in the PEST control file, PEST will run the model once before ceasing execution. This may be a more convenient methodology to use where a model has many input files requiring many template files and/or if an objective function value is required.

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 adjusted to achieve an objective function that is low enough for the model to be considered as "calibrated". 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 control file is named *pestcase.pst* and that a copy of this file named *pestcase.pst.keep* has been made. Suppose also that parameter adjustment is actually carried out on super parameters (see section 10.6.3 of part I of this manual), and that *pestcase_svda.pst* is the PEST control file that is employed for this purpose (for which *pestcase.pst* is the corresponding base parameter PEST control file). Upon each occasion of cessation of *pestcase_svda.pst* execution, optimised base parameter values reside in file *pestcase.bpa*. However the run record file is named *pestcase_svda.rec*.

The following batch file can be used to undertake multiple PEST runs on the basis of starting base parameter values 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 pestcase.pst.keep pestcase.pst
pest pestcase_svda
find /I "optimised mea" pestcase_svda.rec >> record.dat
copy pestcase_svda.rec pestcase_svda.rec.%%i
copy pestcase.bpa pestcase.bpa.%%i)
```

Figure 12.3 A batch file used for conducting a sequence of SVD-assisted parameter estimation runs.

After running the batch file depicted in figure 12.3, the sequence of resulting run record files will be named *pestcase_svda.rec.1*, *pestcase_svda.rec.2*, etc. Respective optimised parameter value files will be named *pestcase.bpa.1*, *pestcase.bpa.2* etc. Note that a "global record file" named *record.dat* is also kept. After each PEST run, the value of the optimised measurement objective function achieved during that run is appended to this file. This provides a continuously-updated record of the success or otherwise of the sequence of PEST runs.

12.8.2 The RDMULRES Input File

RDMULRES requires a control file, an example of which is shown in figure 12.4.

```
# 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
```

Figure 12.4 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 (see below) from being too wide.

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; see chapter 2 of part I of this manual. An instruction file suitable for reading the initial and optimised measurement objective function values from a PEST run record file is illustrated in figure 12.5.

```
pif $
$INITIAL CONDITIONS:$
$measurement$  $=$ !initobj!
$OPTIMISATION RESULTS$
$Optimised measurement$ $=$ !finalobj!
```

Figure 12.5 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 replaced 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.

12.8.3 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 in figure 12.6. Observation names head all columns except the first; 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
```

Figure 12.6 Part of a RDMULRES output file.

A RDMULRES output file is easily imported into a spreadsheet or graphing program for further processing.

12.9 MULPARTAB

12.9.1 General

MULPARTAB stands for "multiple parameter table". It can be used in the postprocessing of multiple PEST runs undertaken, for example, as part of 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 on each occasion (see section 10.6.3 of part I of this manual). 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 succeed in this regard 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.

12.9.2 Using MULPARTAB

MULPARTAB is run using the command

```
mulpartab parfile listfile outfile
```

where

parfile	is a generic parameter value filename,
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 is provided in figure 12.7.

```
# An integer list file

1
2-4
5
7-20
```

Figure 12.7 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 in figure 12.8. As is apparent, the file is comprised of multiple columns. The first of these columns contains parameter names. Columns containing parameter values as read from parameter value files follow this first column. The header for each of these latter columns is the integer index which is included in the filename from which parameter values recorded 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

Figure 12.8 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 you 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 error message to the screen.

12.10 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".)

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; see figure 12.7.

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 an integer from 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.

12.11 COMFILNME1

COMFILNME1 does the same thing as COMFILNME except for the fact that it does not overwrite existing files. Instead it demands that a new filename sequence be created. It is run using the command

```
comflenme datfile listfile newdatfile
```

where

datfile is a generic data or text file name,
listfile contains a list of integer indices, and
newdatfile is a generic data or text file name different from *datfile*.

12.12 REGPRED

12.12.1 General

The purpose of REGPRED is to write a PEST input dataset in which "regularised nonlinear predictive uncertainty analysis" is carried out. REGPRED is an older utility; however it may be useful to some despite the fact that solution of a constrained maximisation/minimisation problem in the highly parameterised context is often a numerically expensive undertaking. Calibration-constrained Monte Carlo methods normally provide better options for post-calibration nonlinear uncertainty analysis where parameter numbers are high and model run times are long.

Use of REGPRED is based on the premise that model calibration has been accomplished through regularised inversion. As is explained by Doherty (2015), 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 dataset; those within the latter subspace are not. The potential errors associated with estimates of parameter combinations associated with the former subspace are a function of measurement noise; this is statistically characterised by the covariance matrix C(**ε**). The potential errors associated with estimates of the latter parameter combinations are a function of the "innate variability" of real-world parameters; this is statistically characterised by the C(**k**) covariance matrix. Both of these matrices are discussed by Doherty (2015) (and referred to extensively in this manual).

REGPRED writes a PEST input dataset in which PEST is asked to run in "predictive analysis" mode. When run in this mode, PEST maximises or minimises a selected 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.

12.12.2 Theory

This subsection echoes some of the theory presented in section 8.4.4 of Doherty (2015). However it is presented herein in a way that is relevant to functionality embodied in the REGPRED utility.

Null Space Constraints

Let $\mathbf{k}$ represent parameters employed by a model. Let $\underline{\mathbf{k}}$ represent the parameter set achieved through model calibration. From equation 5.5.10 of Doherty (2015) the covariance matrix of post-calibration parameter error is calculated as

$$C(\underline{\mathbf{k}} - \mathbf{k}) = (\mathbf{I} - \mathbf{R})C(\mathbf{k})(\mathbf{I} - \mathbf{R})^t + \mathbf{G}C(\boldsymbol{\varepsilon})\mathbf{G}^t \qquad (12.12.1)$$

where

- $\mathbf{k}$ represents true model parameters;
- $\underline{\mathbf{k}}$ represents calibrated model parameters;
- $C(\mathbf{k})$ represents the prior parameter covariance matrix, this describing the innate variability of real-world parameters;
- $C(\boldsymbol{\varepsilon})$ represents the covariance matrix of measurement noise, often assumed to be diagonal;
- $\mathbf{R}$ is the so-called "resolution matrix"; and
- $\mathbf{G}$ is the matrix through which estimated parameter values (i.e. the elements of $\underline{\mathbf{k}}$) are calculated from measurements comprising the calibration dataset (which are denoted by $\mathbf{h}$ in Doherty, 2015); $\mathbf{G}$ is referred to herein as the "parameter solution matrix", or more simply as the "$\mathbf{G}$ matrix".

Meanwhile the actual (and unknowable) errors in the calibrated parameter set $\underline{\mathbf{k}}$ are given by

$$\underline{\mathbf{k}} - \mathbf{k} = -(\mathbf{I} - \mathbf{R})\mathbf{k} + \mathbf{G}\boldsymbol{\varepsilon} \qquad (12.12.2)$$

Where regularised inversion is achieved through singular value decomposition $\mathbf{R}$, $\mathbf{I}$-$\mathbf{R}$ and $\mathbf{G}$ are calculated as

$$\mathbf{R} = \mathbf{V}_1\mathbf{V}^t_1 \qquad (12.12.3)$$

$$\mathbf{I}\text{-}\mathbf{R} = \mathbf{V}_2\mathbf{V}_2^t \qquad (12.12.4)$$

and

$$\mathbf{G} = \mathbf{V}_1\mathbf{S}^{-1}{}_1\mathbf{U}_1{}^t \qquad (12.12.5)$$

where the $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$ matrices are defined through singular value decomposition of the linearized model matrix $\mathbf{Z}$ (calculated as the Jacobian matrix) through

$$\mathbf{J} = \mathbf{USV}^t \qquad (12.12.6)$$

(Normally singular value decomposition is actually undertaken on the weighted Jacobian matrix $\mathbf{Q}^{1/2}\mathbf{J}$ where, ideally, the weight matrix $\mathbf{Q}$ is proportional to $C^{-1}(\boldsymbol{\varepsilon})$. $C(\boldsymbol{\varepsilon})$ in equation 12.12.1 then becomes the identity matrix $\mathbf{I}$.)

The columns of $\mathbf{V}_1$ and $\mathbf{V}_2$ in equations 12.12.3 and 12.12.4 span the calibration solution and null spaces respectively.

To explore the error range of a prediction, the constrained maximisation/minimisation process to be described shortly imposes constraints on $\underline{\mathbf{k}}$-$\mathbf{k}$ as $\mathbf{k}$ is varied in order to maximise or minimise that prediction. It is assumed that a set of calibrated parameters $\underline{\mathbf{k}}$ already exists. The extent to which deviations from this set are tolerable at a certain confidence level is defined by $C(\underline{\mathbf{k}}\text{-}\mathbf{k})$ of equation 12.12.1. Where such deviations are along directions spanned by the calibration null space, tolerance of movement of $\mathbf{k}$ is governed by the first term of equation 12.12.1; the governing probability distribution in this case is $C(\mathbf{k})$ (i.e. expert knowledge). To the extent to which changes in parameter values are incurred along directions spanned by the solution space, the second term of 12.12.1 exerts constraints; for this term the governing probability distribution is $C(\boldsymbol{\varepsilon})$, that is, the stochastic description of measurement noise (of which so-called structural noise arising from model defects is probably a significant contributor – but this is a matter for another time).

REGPRED asks the user to nominate the number of dimensions comprising the calibration solution space. This is the same as the number of columns in $\mathbf{V}_1$ of equation 12.12.3. The SUPCALC utility can provide some assistance in making this determination. It does not have to be exact (and in fact can rarely be exact). REGPRED actually undertakes singular value decomposition of the resolution matrix $\mathbf{R}$ read from a file produced by the RESWRIT utility. In doing so, it calculates the matrices $\mathbf{D}$, $\mathbf{E}$ and $\mathbf{F}$ such that

$$\mathbf{R}=\mathbf{D}\mathbf{E}\mathbf{F}^t \tag{12.12.7}$$

For the set of parameter values $\mathbf{k}$ employed in the predictive analysis maximisation/minimisation process which it sets up, REGPRED computes

$$\mathbf{g} = \mathbf{D}_2^t(\mathbf{k} - \underline{\mathbf{k}}) \tag{12.12.8}$$

where $\mathbf{D}_2$ is comprised of the last m-k columns of $\mathbf{D}$, k being the number of dimensions in the solution space as provided by the user, and m being the total number of parameters featured in the inversion process. $\mathbf{D}_2$ approximately spans the calibration null space. The $\mathbf{g}$ vector has m-k elements.

From (12.12.2)

$$\mathbf{g} = \mathbf{D}_2^t(\underline{\mathbf{k}} - \mathbf{k}) = -\mathbf{D}_2^t(\mathbf{I} - \mathbf{R})\mathbf{k} + \mathbf{D}_2^t\mathbf{G}\boldsymbol{\varepsilon} \tag{12.12.9}$$

The second term on the right side of equation 12.12.9 is assumed to be zero; it is in fact exactly zero where regularisation in undertaken using singular value decomposition. Hence the covariance matrix of $\mathbf{g}$ is given by

$$C(\mathbf{g})= \mathbf{D}_2^tC(\underline{\mathbf{k}} - \mathbf{k})\mathbf{D}_2 = \mathbf{D}_2^t(\mathbf{I} - \mathbf{R})C(\mathbf{k})(\mathbf{I} - \mathbf{R})^t\mathbf{D}_2 \tag{12.12.10}$$

In the predictive analysis PEST control file written by REGPRED, all elements of the $\mathbf{g}$ matrix are read as observations (the calculation of $\mathbf{g}$ through equation 12.12.8 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 $C(\mathbf{g})$ which is computed by REGPRED using equation 12.12.10 and written to a covariance matrix file ready for the use of PEST. Because the elements of $\mathbf{g}$ thus comprise part of the observation dataset, deviations of $\mathbf{g}$ 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 employed as a solution mechanism for the inverse problem) contains the **V** matrix of equation 12.12.6, provided the EIGWRITE variable in the "singular value decomposition" section of the pertinent PEST control file was 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 $\mathbf{D}_2^t$ and $\mathbf{D}_2^t$ **(I – R)** in the above procedure. Where regularised inversion was implemented using singular value decomposition, and thus **(I - R)** is equal to $\mathbf{V}_2\mathbf{V}_2^t$, all of these are equivalent.

Solution Space Constraints

Error in the estimation of those combinations of parameters that lie within the calibration solution space arises solely from measurement (and structural) noise, the stochastic properties of which are supposedly described by the C(**ε**) covariance matrix appearing in equation 12.12.1.

Suppose that the estimated parameter set $\underline{\mathbf{k}}$ 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 $\underline{\mathbf{k}}+\delta\underline{\mathbf{k}}$ 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 as assessed 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 $\delta\underline{\mathbf{k}}$ to the calibrated parameter set $\underline{\mathbf{k}}$ will be statistically unlikely. As a consequence it is then unlikely that $\underline{\mathbf{k}}+\delta\underline{\mathbf{k}}$ can be considered as an alternative to $\underline{\mathbf{k}}$ at the same significance level; hence $\delta\underline{\mathbf{k}}$ 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 matrix 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 $\mathbf{o}+\boldsymbol{\varepsilon}_1$. Implied in the calibration process, and the assumption of truly random measurement noise, is the assumption that a model can perfectly replicate true system behaviour if it is provided with the correct parameter set. Suppose that this parameter set, after projection onto the calibration solution space, is $\underline{\mathbf{k}}$. Now if any parameter set $\underline{\mathbf{k}}+\delta\underline{\mathbf{k}}$ gives rise to a set of model outputs $\mathbf{o}+\delta\underline{\mathbf{o}}$ to which the above measurement noise $\boldsymbol{\varepsilon}_1$ is added, the resulting model outputs will be $\mathbf{h}+\delta\underline{\mathbf{o}}$. If $\delta\underline{\mathbf{o}}$ is a statistically unlikely random variable as assessed using the covariance matrix C(**ε**), then $\mathbf{h}+\delta\underline{\mathbf{o}}$ is significantly different from **h** and the parameter difference $\delta\underline{\mathbf{k}}$ thus introduces discernable differences to the calibration dataset $\underline{\mathbf{k}}$, notwithstanding the noise content of this data. $\underline{\mathbf{k}}+\delta\underline{\mathbf{k}}$ can thus *not* be considered as a parameter set which "calibrates" the model at a certain significance level because it results in a statistically unlikely difference in model outputs, at that same significance level.

Following this logic, when implementing maximisation/minimisation 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 maximisation/minimisation 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.

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 you have 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 12.12.8. It is assumed that you have already calculated this resolution matrix.

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; see section 2.5 of this manual for specifications of this file type. (It should be noted here that observation weights employed in the predictive analysis process can differ from those employed in the parameter estimation process. REGPREG, however, does not allow a covariance matrix to be used to characterise the uncertainties of field measurements.)

The observation dataset written to the new PEST control file is supplemented by the elements of **g**; 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 equation 12.12.10 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 on the one hand and parameter differences projected onto the calibration null space on the other hand. 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 accord with their statistical distributions. If the noise associated with measurements is Gaussian, and if C(**k**) is multiGaussian, the square root of the objective function thus measures, in normalised observation space, the distance of departure of the collective model outputs 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 (see section 8.3 of Doherty (2015) and documentation of the ASSESPAR program for further details). 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 zero is 3 standard deviations. Thus at this point the "observations" are statistically different from their optimised counterparts at the 99.7% significance level. Given the definition of these observations, this means that there is a very low likelihood that the corresponding parameter set satisfies the stochastic requirements of C(**k**), and that model output deviations calculated on the basis of the corresponding parameter set satisfy the stochastic requirements of C(**ε**); thus any prediction made on the basis of that parameter set is equally unlikely.

12.12.3 Preparing for a REGPRED Run

Obtaining the Resolution Matrix

Prior to running REGPRED, a regularised inversion exercise must have been carried out. Regularisation could have been achieved using singular value decomposition alone (in which case PEST was run in "estimation" mode), and/or using Tikhonov constraints (in which case PEST was run in "regularisation" mode"), and/or using SVD-assist (in either mode). 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 a file of its own using PEST matrix file format (see section 2.4 of this manual). 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.

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). The second of the above tasks can be carried out using the OBSREP utility.

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, the (measurement) objective function should be 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 interest. 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.

12.12.4 Running REGPRED

REGPRED's 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 (containing 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 estimated parameter values (and optimised model outputs) were obtained.

REGPRED next asks

```
Enter dimensions of solution space:
```

This question may sometimes be difficult to answer. The SUPCALC utility can help.

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 for the inverse problem. If the EIGWRITE variable was set to 1 on that run, the **V** matrix comprised of eigenvectors of $\mathbf{Q}^{1/2}\mathbf{J}$ is recorded during every iteration in this file. REGPRED next asks

```
Get SVD for which iteration number? <Enter if 1>:
```

If the singular value decomposition run was undertaken for the purposes of model calibration, enter the number of the last iteration undertaken. However if PEST was only run for one 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 singular value decomposition had not been used as a solution

device for the inverse problem. Note also that, ideally, the RLAMBDA1 control variable should be set to zero if PEST is run purely for the sake of obtaining this file, for then the contents of this file will not reflect the value of the Marquardt lambda.

REGPRED's next prompt is

```
Enter dimensions of solution space:
```

See above.

Next, whether REGPRED reads a resolution matrix file or a SVD file, it prompts for the name of a parameter uncertainty file and for the name of an observation uncertainty file.

```
Enter name of parameter uncertainty file:
Enter name of observation uncertainty file:
```

See section 2.5 of this manual for specifications of an uncertainty file. In the present context the purpose of the above files is to supply the C(**k**) and C(**ε**) matrices described above. Note that if parameters are log-transformed through the parameter estimation process, then pertinent elements of C(**k**) 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 as that 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 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 chapter 8 of part I of this manual, PD0 is the objective function constraint imposed through the predictive 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 is 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 chapter 8 of part I of this manual. As explained in that chapter, 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.

The Modified Model

The model run by PEST as part of the predictive analysis process must compute **g** of equation 12.12.9 on the basis of current 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 parameter values **k**; 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 **k** and optimised parameter values $\underline{\mathbf{k}}$. 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 $\mathbf{k}-\underline{\mathbf{k}}$ vector is written to a file named *pardiff###.mat*.

The MATPROD utility is used to compute **g** using equation 12.12.9. $\mathbf{D}_2^t$ (or $\mathbf{V}_2^t$ if the singular value decomposition option is employed) 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 equation 12.12.10) 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 or BEOPEST, it is important to ensure that these executables are transferred to agent machines.

12.12.5 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, REGPRED will prompt (as discussed 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 you do not want this, REGPRED will ask you 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 your 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 your responsibility to undertake this task. (REGPRED reminds you 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 you from being severely disappointed if, on returning to your computer the morning after setting up PEST to run all night, you find 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 you 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, you should alter them yourself.

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. There may be an occasion or two when PEST will 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.

12.12.6 Using the SVD Matrix Option

If you wish to provide REGPRED with an SVD file instead of a resolution data file and you don't already have one, then follow these steps. It is assumed that you have a PEST control

file in which optimised parameter values are initial values, and that you have run PEST to produce a corresponding JCO file.

1. Set NUMLAM to 1 and RLAMBDA1 to zero in the PEST control file.
2. Remove all regularisation observations and prior information from this file; this can be facilitated using the SUBREG1 utility.
3. Add a "singular value decomposition" section to this file. Be sure to set EIGWRITE to 1.
4. Set NOPTMAX to 1.
5. Use the JCO2JCO utility to create a JCO file corresponding to this new PEST control file.
6. 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 that was just written by JCO2JCO. PEST will then undertake three model runs – one to compute the objective function, one to test an upgrade vector, and one on the basis of "best fit parameters". However no model runs will be required for calculation of the Jacobian matrix, as this is read from the existing JCO file.

12.13 ASSESSPAR

12.13.1 General

The ASSESSPAR utility can be used to assist in model-based hypothesis-testing.

Suppose that a model has been calibrated. Suppose that it is then re-calibrated with an extra "observation" included in the calibration process, this being the value of a prediction of interest. (This is effectively done when using PEST in "pareto" mode.) If parameter values that emerge from this second calibration process are "unreasonable", then the prediction can be deemed to be unlikely.

The integrity of model-based hypothesis-testing conducted in this way rests upon the model's ability to allow predictive values to be computed if, indeed, these values are compatible with an expert-knowledge-based assessment of hydraulic properties. In the groundwater modelling context, this will normally require that calibration be undertaken using highly-parameterized inversion, so that predictive possibilities are not artificially precluded by a parameterization scheme that is incapable of representing geologically realistic heterogeneity. Both the first and second calibration exercises discussed above will therefore probably employ Tikhonov regularisation in which preferred values are assigned to parameters through regularisation prior information. A PEST control file will probably therefore exist in which "expected parameter values" from a pre-calibration (or pre-re-calibration) point of view are represented as initial values within this file. Assessment of parameter sets which emerge from the hypothesis-testing calibration process is then based on differences between calibrated parameter values and these prior expected parameter values. ASSESSPAR allows this comparison to be made on a statistical basis.

ASSESSPAR computes two variates, one a chi-square variate and one a normal variate. (Here it is assumed that prior parameter probabilities are multiGaussian in nature.) Optionally, a prediction-specific normal variate can be computed based on sensitivities of a prediction of interest to model parameters. If this is to be done, the model calibration dataset will need to include this prediction so that the sensitivity of this prediction to model

parameters is available from a JCO file.

If desired, other statistics can be calculated on the basis of information recorded in the ASSESSPAR output file. This file tabulates the values of "eigenparameters"; the standard deviation of each of these is 1.0, while each is statistically independent of other eigenparameters. If a Gaussian prior parameter distribution is assumed, parameter (and hence predictive) confidence intervals are readily calculated from the values of these eigenparameters.

12.13.2 Theory

The theory presented in this sub-section is taken from sections 8.3 and 8.5 of Doherty (2015). It is repeated here in a way that allows operation of the ASSESSPAR utility to be better understood.

Let **k** denote a parameter set. Actually, let it denote the differences between an actual parameter set and a "base" parameter set whose values are prior expected parameter values. Thus the expected value of the **k** parameter vector is the vector **0**.

Let the covariance matrix of **k** be denoted as C(**k**). Because C(**k**) must be a symmetric, positive-definite matrix, it can be written (through singular value decomposition) as

$$C(\mathbf{k}) = \mathbf{E}\mathbf{F}\mathbf{E}^t \tag{12.13.1}$$

where **F** is a diagonal matrix with positive diagonal elements.

Let the "eigenparameter set" **m** be defined through the following transformation (which is the same as the Kahunen-Loève transformation discussed by Doherty, 2015).

$$\mathbf{m} = \mathbf{F}^{-1/2}\mathbf{E}^t\mathbf{k} \tag{12.13.2}$$

It is easily verified that the covariance matrix of **m**, i.e. C(**m**), is the identity matrix **I**. Hence, by definition, the scalar variable $\mathbf{m}^t\mathbf{m}$ has a chi-square distribution with m degrees of freedom, where m is the number of elements of **k** and hence of **m**. Basic matrix manipulation demonstrates that

$$\mathbf{m}^t\mathbf{m} = \mathbf{k}^t\mathbf{E}\mathbf{F}^{-1/2}\mathbf{F}^{-1/2}\mathbf{E}^t\mathbf{k} = \mathbf{k}^t\mathbf{E}\mathbf{F}^{-1}\mathbf{E}^t\mathbf{k} = \mathbf{k}^t\mathbf{C}^{-1}(\mathbf{k})\mathbf{k} \tag{12.13.3}$$

Hence $\mathbf{k}^t\mathbf{C}^{-1}(\mathbf{k})\mathbf{k}$ also has a chi-square distribution. Tables of probabilities associated with chi-square values are readily available; calculators are also available on the internet. Note that as the number of degrees of freedom increases, the chi-square distribution approaches a normal distribution with mean m and variance $2m$. This relationship can be used to assess parameter credibility in highly parameterized cases as standard tables do not usually extend beyond m values of about 100.

Let t be a scalar that is calculated from **m** (defined as above) through use of a vector **j** as

$$t = \mathbf{j}^t\mathbf{m} \tag{12.13.4}$$

Through the basic propagation of variance relationship, the variance of t is given by:

$$\sigma^2_t = \mathbf{j}^t\mathbf{C}(\mathbf{m})\mathbf{j} = \mathbf{j}^t\mathbf{j} \tag{12.13.5}$$

If follows that if **j** is a unit vector then the variance (and hence standard deviation) of **j** is 1.0. If C(**m**) pertains to a normal distribution, then this provides us with a normal variate through which parameter credibility can be assessed. Useful options for **j** may include vectors for which all elements except that pertaining to a parameter of interest are zero while that pertaining to the parameter of interest is 1; this allows the credibility of the value of an

individual parameter to be assessed.

For predictive credibility assessment a more useful unit vector may be

$$\mathbf{j} = \mathbf{v} = \mathbf{w}/(\mathbf{w}^t\mathbf{w})^{1/2} \quad (12.13.6)$$

where **w** encompasses sensitivities of a prediction of interest to elements of the eigenparameter set **m**.

Suppose that the sensitivity of a prediction *s* to the real parameter set **k** is given by the vector **y**. Then

$$s = \mathbf{y}^t\mathbf{k} \quad (12.13.7)$$

If **w** is calculated as

$$\mathbf{w} = \mathbf{F}^{1/2}\mathbf{E}^t\mathbf{y} \quad (12.13.8)$$

then

$$\mathbf{w}^t\mathbf{m} = \mathbf{y}^t\mathbf{E}\mathbf{F}^{1/2}\mathbf{F}^{-1/2}\mathbf{E}^t\mathbf{k} = \mathbf{y}^t\mathbf{k} = s \quad (12.13.9)$$

the second last equality following from orthonormality of **E**. Because **v** is a unit vector, it follows that the scalar

$$s' = \mathbf{v}^t\mathbf{m} = \mathbf{w}^t\mathbf{m}/(\mathbf{w}^t\mathbf{w})^{1/2} = \mathbf{y}^t\mathbf{k}/(\mathbf{w}^t\mathbf{w})^{1/2} \quad (12.13.10)$$

has a standard deviation of 1. From (12.12.8) it follows that

$$\mathbf{w}^t\mathbf{w} = \mathbf{y}^t\mathbf{E}\mathbf{F}^{1/2}\mathbf{F}^{1/2}\mathbf{E}^t\mathbf{y} = \mathbf{y}^t\mathrm{C}(\mathbf{k})\mathbf{y} \quad (12.13.11)$$

Hence

$$s' = \mathbf{y}^t\mathbf{k}/[\mathbf{y}^t\mathrm{C}(\mathbf{k})\mathbf{y}]^{1/2} \quad (12.13.12)$$

has a standard deviation of 1. If C(**k**) pertains to a multi-Gaussian distribution, then *s′* is a standard normal variate. Its value is then easily employed for assessing parameter confidence in contexts such as that described above where parameter values are "calibrated" to ensure the occurrence of a prediction *s*. In doing so, predictive confidence is also assessed. Calibration to ensure the occurrence of a prediction can be achieved through a single calibration exercise in which the prediction is included in the calibration dataset; alternatively it can be achieved through using PEST in "pareto" mode.

12.13.3 Using ASSESSPAR

ASSESSPAR is run using the command

```
assesspar parfile pestfile uncfile outfile [senfile]
```

where

parfile	is a parameter value file,
pestfile	is a PEST control file containing prior expected parameter values,
uncfile	is a parameter uncertainty file,
outfile	is the eigenparameter output file, and
senfile	is an optional predictive sensitivity file.

ASSESSPAR assumes that expected parameter values are housed as initial values in the user-nominated PEST control file. The transformation status of parameters is also read from this file. As is normal practice, parameter sensitivities and uncertainties pertaining to parameters must pertain to the logs of parameters where parameters are denoted as log-transformed in the

PEST control file.

If a predictive sensitivity file is not supplied, then the normal variate s' of equation 12.13.12 is not calculated. Only the chi-square variable $\mathbf{k}^t\mathbf{C}^{-1}(\mathbf{k})\mathbf{k}$ is calculated in this case. Inversion of the C(**k**) matrix is undertaken using SVD. This may take a while where the matrix is large; the 64bit version of ASSESSPAR may then be required.

Whether or not a predictive sensitivity vector is supplied, the output file written by ASSESSPAR lists all eigenparameters. If numerical singular value decomposition of C(**k**) is undertaken (which occurs if C(**k**) has any off-diagonal elements), these are listed in order of decreasing singular value. Otherwise they are listed in the same order as parameters in the PEST control file as there is then a one-to-one relationship between actual parameters and eigenparameters. Eigenparameters are the elements of the vector **m** of equation 12.13.2. Where C(**k**) pertains to a Gaussian distribution, then each of these are standard normal variates whose credibility is thereby easily assessed.

The values of the chi-square variable and (when the predictive sensitivity file is supplied) the values of the prediction-specific normal variate are each written to the screen and recorded in the ASSESSPAR output file. Note that the degrees of freedom associated with the chi-square distribution is the number of adjustable parameters contained in the PEST control file. This is also listed to the screen and to the ASSESSPAR output file.

Note also that a predictive sensitivity file is easily obtained from a PEST Jacobian matrix file using the JROW2VEC utility.

13. Latin Hypercube Sampling

I would like to thank Sandia National Laboratories for funding the writing of the software documented in the present chapter. On a personal basis, thanks also go to Scott James, Bill Arnold (from SNL) and Paul Reimus (from LANL).

13.1 Introduction

13.1.1 General

This chapter documents three utility programs which allow data interchange between PEST and the LHS program developed by Sandia National Laboratories. Use of these programs can facilitate calibration-constrained uncertainty analysis through combining the sampling efficiencies of the Latin hypercube scheme with the ability of PEST to modify randomly-generated parameter sets in order to allow model outputs to match field data.

A WINDOWS-compiled version of the LHS program named *runlhs.exe* is provided with PEST. It is run by typing its name at the screen prompt, followed by the name of its input file. Preparation of its input file can be undertaken using a standard text editor, or by translation from a PEST input dataset. The LHS software manual is also provided with PEST. It describes the algorithmic details of LHS, and provides specifications for its input file.

The full reference for the LHS software manual is as follows.

Swiler, L.P. and Wyss, G.D., 2004. A User's Guide to Sandia's Latin Hypercube Sampling Software: LHS UNIX Library/Standalone Version. Sandia National Laboratories, Report SAND2004-2439.

13.1.2 Using PEST with LHS

While the utility programs described in the present chapter facilitate use of PEST with LHS, you will nevertheless be required to undertake a certain amount of PEST setup yourself when using these two packages together. Hopefully, however, the task of passing parameter values between them is made easier through use of these programs.

Two possible strategies for joint use of PEST and LHS are now discussed. Other possibilities are left to the user's imagination.

Calibration-Constrained Random Parameter Fields – Method 1

The aim of all calibration-constrained random parameter field generation is to sample the posterior probability distribution of a parameter set. Strategies such as Markov chain Monte Carlo can achieve this; however they may not be useable in contexts where model run times are large. Hence approximate and more model-run-efficient methods may need to be used instead. One such method is the null space Monte Carlo (NSMC) method available through the PEST suite. Ideally, it should be possible to realise further efficiencies through combining this with the Latin hypercube sampling methodology.

A possible strategy for combining NSMC and LHS is as follows.

1. Calibrate a model.
2. Construct a PEST input dataset in which calibrated parameters feature as initial parameter values.

3. Run PEST to obtain a Jacobian matrix based on calibrated parameter values.
4. Generate parameter samples based on the prior probability distribution and prior parameter values using LHS.
5. Use the PNULPAR utility to obtain a secondary set of samples that, through differencing with the calibrated parameter set and null space projection, should "almost calibrate" the model, while retaining null space randomness.
6. Undertake a series of re-calibration operations using SVD-assist as per the NSMC methodology. Recall from section 10.6.3 of part I of this manual that NSMC achieves computational efficiency through
    a. differencing a random parameter field with a calibrated parameter field and removal of any solution space parameter components of this difference which compromise goodness of fit between model outputs and field data (achieved using PNULPAR, as stated above);
    b. use of a limited number of super parameters for parameter adjustment;
    c. re-use of a global set of super parameter sensitivities for the initial re-calibration iteration as applied to all random, null-space-projected parameter sets.

If desired, Tikhonov constraints could be introduced to the suite of PEST runs required to lower the calibration objective function to a suitably low level. These could constrain parameter sets to deviate minimally from their initial LHS-sampled and null space projected values in achieving the desired objective function.

Calibration-Constrained Random Parameter Fields – Method 2

In the above methodology, the LHS method is employed for sampling of the prior parameter probability distribution. Calibration constraints are then applied to these samples through removal of solution space components that compromise goodness of model-to-measurement fit, while retaining null space components which (by definition if the model is linear) do not affect that fit. An alternative strategy is to sample the posterior parameter probability distribution (or an approximation to it) using the LHS sampling methodology. A possible strategy is as follows.

1. Calibrate a model.
2. Construct a PEST input dataset in which calibrated parameters feature as initial parameter values.
3. Run PEST to obtain a Jacobian matrix based on calibrated parameter values.
4. Use the PEST PREDUNC7 utility to obtain a linear approximation to the posterior parameter covariance matrix.
5. Use the PEST MATSVD utility to obtain eigencomponents of this covariance matrix.
6. Using these eigencomponents, define a set of independent parameters of unity variance (see below).
7. Sample these independent parameters using LHS and then back-transform these samples to obtain samples of native parameter values (see below).

8. (Perhaps) use PNULPAR to effect further removal of unwanted solution space components from the above-generated random parameter sets.
9. Using the NSMC method as described above, alter these parameter fields such that they respect calibration constraints.
10. Alternatively, if the dimensions of the null space are not very large, adjust each of these parameter fields in order to lower the objective function by running PEST with the "/i" switch. The first (and possibly only) iteration required to lower the objective function to a suitable level is then obtained "for free", using the same, already-computed, Jacobian matrix calculated using the calibrated parameter set.

Transformation of parameters to achieve statistical independence is now briefly described.

Suppose that the parameter set **k** has a covariance matrix C(**k**). Undertake singular value decomposition of this covariance matrix to obtain matrices **E** and **F** such that

$$C(\mathbf{k}) = \mathbf{EFE}^t \qquad (13.1.1)$$

In equation 13.1.1 **F** is a diagonal matrix containing the eigenvalues of C(**k**) arranged from highest to lowest while **E** is an orthonormal matrix, i.e. a matrix whose columns are orthogonal unit vectors. Using standard propagation of covariance relationships it is easily shown that the vector **m** defined through transformation of **k** as

$$\mathbf{m} = \mathbf{F}^{-1/2}\mathbf{E}^t\mathbf{k} \qquad (13.1.2)$$

is comprised of elements m_i that are statistically independent and that have unit standard deviation. If the m_i are individually sampled using LHS, an equivalent set of **k** parameters can be calculated from thus-sampled **m** vectors through the relationship

$$\mathbf{k} = \mathbf{EF}^{1/2}\mathbf{m} \qquad (13.1.3)$$

In implementing equations 13.1.2 and 13.1.3, the expected values of parameters should be subtracted from the actual parameter values before transformation, and then added back after transformation.

13.2 PHISTATS

Although it was written to expedite use of the Sandia LHS program with PEST, use of PHISTATS is broader than this. It can be used on its own, or as part of any batch file that undertakes repeated PEST runs. It provides a mechanism for an up-to-date reporting of the status of those runs, and of their ability to lower (or not) the objective function.

PHISTATS is run using the command

```
phistats recfile N
```

where *recfile* is the name of a PEST run record file, and *N* is any integer supplied by the user. If run inside a loop recorded in a batch file, *N* will probably be the index of the loop control variable which, under normal operation, will be incremented after each pass through the loop.

When executed in the above manner, PHISTATS reads the nominated run record file. Note that an extension of "*.rec*" is presumed; if this is omitted it will be added to the user-supplied filename automatically. PHISTATS then writes the following information to the screen.

- initial objective function;
- initial contributions to the objective function by all observation groups;

- initial measurement and regularisation objective functions (if PEST is run in "regularisation" mode);
- initial predictive and predictive error terms (if PEST is run in "predictive analysis" mode).
- all of the above statistics as recorded at completion of PEST execution.

Note that PHISTATS will cease execution with an error message if it is asked to read a run record file that documents the outcomes of a PEST run in which PEST is run in "pareto" mode.

Not also that the integer *N* supplied on the command line does not affect PHISTATS's reading of the run record file, or which file it reads. This index is, however, reported to the screen together with the objective function information that it provides. The user is therefore informed of the current loop index.

13.3 LHS2PEST

13.3.1 General

LHS2PEST provides a linkage between PEST and the LHS program written by Sandia National Laboratories. It facilitates the undertaking of multiple PEST runs, each based on a different set of Latin hypercube parameter value samples generated by LHS.

PEST runs can be undertaken for a number of purposes. If the NOPTMAX control variable is set to 0, then PEST will simply run the model once, calculate the objective function and different components thereof, calculate and record some statistics, and then cease execution. If NOPTMAX is set to -1 or -2, PEST will calculate the Jacobian matrix, this comprising sensitivities of members of the observation dataset to the different parameters. If NOPTMAX is set to a positive number, then PEST will undertake parameter estimation; for each parameter estimation run, initial parameter values can thus be Latin hypercube samples of the various parameters.

For NOPTMAX not set to zero, the following should be noted.

- If a model is linear with respect to its parameters, sensitivities recorded in the Jacobian matrix are independent of parameter values.
- If undertaking repeated parameter estimation based on different LHS-generated initial parameter values, and this process does not employ the SVD-assist methodology (do not confuse this with singular value decomposition as a solution device for the inverse problem), consider running PEST with the "/i" switch so that it can re-use the same Jacobian matrix on its first iteration for each of these parameter estimation exercises. Presumably this matrix will have been calculated using a representative set of parameter values - ideally prior expected parameter values. (Note that in the batch file written by LHS2PEST - see below - the command to run PEST is not accompanied by this switch; you must add it to the pertinent command yourself.)
- Through use of the ADDREG1 utility, Tikhonov constraints can be implemented which promulgate maximum adherence of adjusted parameter values to initial parameter values; maximum respect for LHS samples as parameters are adjusted to satisfy calibration constraints is thereby maintained.

Use of LHS2PEST is predicated on the assumption that a PEST input dataset and corresponding LHS input dataset exist. It is also assumed that the same parameters have the same names in both of these datasets. (As will be discussed, either of the files comprising these dataset can possess other parameters as well.) Adherence to this protocol will require that parameter name lengths be restricted to 12 characters or less in the LHS input dataset, as this is the character length limit for a parameter name employed by PEST.

It is further assumed that LHS has been run, and that an LHS output file with parameter sample values has thus been recorded. LHS2PEST reads these samples, and builds a set of PEST parameter values files, each recording the values associated with one sample set. (Parameter value files are described in section 2.2 of this manual.) These files can be used in similar ways to those written by the RANDPAR utility. In particular, through sequential use of the PARREP utility, parameter sets contained in these files can be used as initial values in a sequence of PEST control files. PEST can then use these control files to undertake sequential runs for any of the reasons discussed above.

The tasks performed by LHS2PEST depend on the command that is used to run it. The commands, and the tasks that correspond to these commands, are now described.

13.3.2 LHS2PEST - Level 1

The simplest way to run LHS2PEST is to invoke the command

```
lhs2pest lhsoutfile parfilebase
```

where

lhsoutfile	is the name of an LHS output file that includes parameter sample values, and
parfilebase	is the filename base of a set of PEST parameter values files.

If run using the above command, LHS2PEST does the following.

- It reads the names of parameters which are featured in the LHS output file. If any of these names are greater than 12 characters in length, LHS2PEST reports this as an error and then ceases execution.
- It reads the Latin hypercube samples associated with these parameters.
- For each Latin hypercube sample set, it writes a single PEST parameter value file which contains the values of these samples.

Suppose that *parfilebase* is supplied as "*random*". Then the parameter value files written by PEST are named *random1.par*, *random2.par*, etc. LHS2PEST writes as many of these files as there are sample sets in the LHS output file. Sequence numbers are preserved in writing these files. Hence, in the above example, file *randomN.par* contains values pertaining to the *N*'th Latin hypercube sample generated by LHS.

The following protocols are adopted in writing these parameter value files.

- All parameters are given a SCALE of 1 and an OFFSET of 0.
- The PRECIS and DPOINT control variables are given values of "single" and "point" respectively.

It is important to note that while adoption of a SCALE of 1 and an OFFSET of 0 simplifies the running of LHS2PEST, it may create problems for users who create a PEST control file which employs a different SCALE and OFFSET for some parameters. In this case, parameter

value files may be more useful if they list pre-transformed parameter values rather than actual parameter values so that correct parameter values will be written to model input files after the desired SCALE and OFFSET have been applied. Users to which this applies need to be aware of this situation.

13.3.3 LHS2PEST - Level 2

When run using the command

```
lhs2pest lhsoutfile parfilebase pestfile
```

where *pestfile* is the name of a PEST control file, LHS2PEST reconciles parameters represented within a PEST control file with those represented in the LHS output file. In particular, it informs the user if any parameters cited in the LHS output file are not cited in the PEST control file; it also informs the user if any parameters cited in the PEST control file are not cited in the LHS output file.

When recording parameter value files LHS2PEST adopts the following protocols when run in the above way.

- The only parameters which are recorded in these files are those which are featured in the PEST control file. Hence a parameter that is cited in the LHS output file but is not cited in the PEST control file is not recorded in the parameter value files written by LHS2PEST.
- Where a parameter is cited in the PEST control file, but not featured in the LHS output file, it is provided with the same value in all parameter value files, this being its initial value as recorded in the PEST control file.
- All parameters which feature in both PEST control and LHS output files are given Latin hypercube sample values as recorded in the LHS output file. Note that this occurs regardless of whether they are adjustable, fixed or tied in the PEST control file. Provide them with alternative names in the latter file if you do not want this to happen.
- If an LHS-generated sample for a particular parameter has a value which exceeds the upper bound for that parameter as recorded on the PEST control file, or is lower than that parameter's lower bound, then you will be notified of this. At the same time the sampled parameter value is clipped to respect these bounds.
- If a parameter is featured in the PEST control file, but not in the LHS output file, and if that parameter is tied to a parameter whose value is informed by the LHS output file, the value of the tied parameter is adjusted in accordance with the LHS-supplied value for its parent parameter. If, in doing this, its upper or lower bound is transgressed, it is adjusted back to its upper or lower bound and a warning is issued.
- The SCALE and OFFSET value provided to each parameter in LHS-recorded parameter value files is that with which it is provided in the PEST control file.
- Values for PEST control variables PRECIS and DPOINT recorded in the headers to parameter value files are equivalent to those used in the PEST control file read by LHS2PEST.

Parameter value files written by LHS2PEST when run in level 2 mode are thus perfectly compatible with the PEST control file cited on the LHS2PEST command line.

13.3.4 LHS2PEST - Level 3

To activate LHS2PEST level 3 functionality, it must be run using the following command.

```
lhs2pest lhsoutfile parfilebase pestfile batchfile
```

In this case LHS2PEST does everything that level 2 functionality requires. However it also writes a batch file through which PEST can be run repeatedly based on the parameter value file sequence that LHS2PEST also generates. An example batch file is shown below.

```
@echo off
rem ############################################################
rem Delete an existing record file.
rem ############################################################

del /P record.dat
echo > record.dat

rem ############################################################
rem Do all the PEST runs.
rem ############################################################

for /L %%i in (1,1,100) do (
parrep random%%i.par temp1.pst t###.pst
del t###.rec
pest t###.pst
copy t###.res t###%%i.res
phistats t###.rec %%i >> record.dat
)
```

Figure 13.1 A batch file written by LHS2PEST.

The batch file written by LHS2PEST runs PEST repeatedly through a processing loop. This loop is traversed as many times are there are Latin hypercube samples provided in the LHS output file. On each occasion on which PEST is run, the PARREP utility is first run in order to write a new PEST control file which is identical to the PEST control file nominated on the LHS2PEST command line except for the fact that its initial parameter values are those provided by the respective parameter value file, and hence constitute LHS-generated Latin hypercube parameter sample values. This PEST control file is named *t###.pst*. After each PEST run, the PEST residuals file (named *t###.res*) is copied to file *t###N.res*) where *N* is the run index. Hence you can inspect the details of model-to-measurement fit calculated on the basis of all Latin hypercube parameter sets. If you do not want these files saved, comment out or delete the respective line from the batch file. Alternatively, request that other PEST files be saved by adding other commands of a similar type within the processing loop provided in the batch file.

Each PEST run is followed by a PHISTATS run. This provides you with the opportunity to inspect a file named *record.dat* (rename this file in the batch file if you wish) at any loop count, and at the end of the loop altogether when all PEST runs have been completed. This file lists the sequence of objective functions that have been calculated on the basis of the different Latin hypercube sample sets that have been used until the time of inspection. At the start of the loop an existing file of this name (i.e. *record.dat*) is deleted. The batch file prompts for permission to delete this file before it actually does this.

When run in level 3 mode, LHS2PEST writes the following message to the screen just before it terminates execution.

```
Note:-
The value for NOPTMAX for all PEST runs initiated through the batch file
will be 0.

If this is not your intention, alter NOPTMAX to an appropriate value in the
PEST control file supplied in the command line.
```

Figure 13.2 Warning written to screen by LHS2PEST on cessation of level 3 execution.

See the discussion of the NOPTMAX variable earlier in this section (and in part I of this manual). NOPTMAX is an integer; it is the first variable on the 9th line of the PEST control file.

13.4 PEST2LHS

13.4.1 General

PEST2LHS builds an input file for the LHS program using information contained within a PEST control file, possibly supplemented by a PEST-compatible parameter uncertainty file (see section 2.5 of this manual for specifications of this file type). The LHS program can then be run to generate a suite of Latin hypercube samples of parameters featured in the original PEST control file. These samples can then be used by PEST for calculation of objective functions, or for more complex tasks such as model calibration based on these samples as initial values. (Creation of a PEST input dataset based on LHS-generated samples can be accomplished using the LHS2PEST program described above.)

The LHS input file written by PEST2LHS is simple. Of the many options that LHS provides for its input dataset, PEST2LHS supports only a few. The user is advised to alter the PEST2LHS-generated LHS input file him/herself to request options not supported by PEST2LHS if these are required. Because it supports only a few LHS options and employs many defaults, use of PEST2LHS is relatively easy. Adoption of simplicity in its design specifications, accompanied by a recommendation for the user to directly edit the LHS input file written by PEST2LHS him/herself to gain access to more complex LHS functionality, is based on the premise that provision of a complex input dataset to PEST2LHS would be just as difficult (if not more difficult) than supplying a complex input dataset to LHS directly through editing of the latter's input file.

13.4.2 Parameter Probability Distributions

PEST-suite programs which require characterization of parameter uncertainty can receive such characterization through a "parameter uncertainty file". As is documented in section 2.5 of this manual, this file can provide standard deviations for individual parameters; it can also reference other files which provide covariance matrices for groups of parameters. PEST2LHS, too, can read a parameter uncertainty file (and covariance matrix files cited therein) in order to obtain specifications for the uncertainties of parameters cited within a PEST control file, these being required for specification of probability distributions on the LHS input file which it writes. However its design is such that if it employs variances and covariances read from a parameter uncertainty file, it simultaneously assumes that all adjustable parameters cited in the PEST control file which it reads are described by either a normal distribution (if they are untransformed) or a log-normal distribution (if they are log-transformed).

If a parameter is log-transformed then the uncertainty information pertaining to that parameter provided in a parameter uncertainty file must pertain to the log to base 10 of that

parameter. As stated above, such information will constitute either the standard deviation of the parameter, or its variance and covariances with other parameters supplied through a covariance matrix file cited in the parameter uncertainty file. To maintain compatibility with other PEST-suite programs, the assumption of base-10 logarithms in the parameter uncertainty file is maintained; PEST2LHS makes the conversion to natural logarithms when writing an LHS input dataset.

If a parameter is cited as untransformed in a PEST control file (i.e. if it has a PARTRANS value of "none" in the "parameter data" section of that control file), then the uncertainty information contained in a parameter uncertainty file, or in any covariance matrices cited in that file, must pertain to the native parameter.

As an alternative to obtaining parameter uncertainty information from a parameter uncertainty file, PEST2LHS can calculate parameter uncertainty statistics itself from parameter bounds provided in the PEST control file. If PEST2LHS is asked to do this, it does not then need to read a parameter uncertainty file at all. Furthermore, if this option for parameter uncertainty characterization is taken, the user can choose between the uniform, triangular and normal probability distributions for expressing that uncertainty. These distributions are assigned on a parameter group by parameter group basis. (Recall that parameter groups are named in the "parameter groups" section of a PEST control file and that individual parameters are assigned to these groups within the "parameter data" section of that file.)

Regardless of whether parameter uncertainties are supplied through an uncertainty file, or are calculated from parameter bounds, PEST2LHS provides you with two options for denoting the parameter value at which the probability distribution is at its peak. The first option is to calculate this parameter value as the midpoint of the interval spanned by a parameter's lower and upper bounds (or the midpoint of the interval spanned by the natural logs of a parameter's lower and upper bounds if the parameter is log-transformed in the PEST control file). The second is to use a parameter's initial value as provided in the PEST control file (or its natural log if the parameter is log-transformed) as the parameter value of peak probability. (Note that this is not applicable to the uniform probability distribution as this distribution has no maximum.)

Where PEST2LHS specifies that a parameter is normally or log-normally distributed when recording its probability distribution specifications on an LHS input file, this distribution is actually specified as a bounded distribution. The bounds that are applied to this distribution are the same as those placed on parameter values in the PEST control file. This ensures that Latin hypercube samples do not transgress these bounds. Where a uniform or triangular probability distribution is employed, respect for parameter bounds is built into the probability distribution specifications themselves.

PEST2LHS assigns probability distributions only to adjustable parameters (i.e. to parameters that are neither fixed nor tied to a parent parameter in the PEST control file which it reads). Tied and fixed parameters are therefore not featured in the LHS input file written by PEST2LHS.

13.4.3 PEST2LHS Usage Details

In contrast to many programs of the PEST suite, PEST2LHS does not receive its usage information through command line arguments. This is because it requires more information than can conveniently be provided through a limited number of such arguments. Hence it prompts the user for the information which it needs.

PEST2LHS commences execution with the prompt

```
Enter name of PEST control file:
```

On having received the name of this file, PEST2LHS checks for its existence, and then reads it. Having done this, it issues the following prompt.

```
There are two options for specifying parameter probability distributions:
   For (log)normal distributions and an uncertainty file           - enter 1
   For user-specified distributions with bounds-derived descriptors - enter 2
Enter your choice:
```

As stated above, PEST2LHS supports two options for specifying parameter uncertainties. The first option is for all of these to be provided through a traditional PEST uncertainty file. The second is for probability distributions to be provided on a parameter group by parameter group basis. As PEST2LHS behaviour is somewhat specific to the selected option, its two different modes of parameter uncertainty specification are now discussed separately.

User-Specified Distributions

Regardless of which option is selected in response to the above prompt, PEST2LHS's next prompt is

```
For maximum value of non-uniform distributions:-
   If halfway between (log) bounds                    - enter 1
   If parameter initial value from PEST control file  - enter 2
 Enter your choice:
```

Complete specification of the triangular and normal distributions requires that a parameter value of maximum probability be specified. In the case of the normal distribution, PEST2LHS can calculate this maximum probability value in either of the two ways specified in the above prompt. In the first case, the peak of the distribution is calculated to correspond to the mean of the parameter's lower and upper bounds as recorded in the PEST control file. In the second case it is associated with a parameter value equal to the initial value for the pertinent parameter as read from the PEST control file. Note that this second option can lead to a non-symmetrical probability distribution.

Where a parameter is log-transformed (which for PEST2LHS is decreed as being incompatible with selection of a triangular distribution as the log-triangular distribution option is not supported by LHS) the parameter abscissa calculated by PEST2LHS for the peak value of what LHS refers to as the "underlying distribution" depends on which of the above options is selected. If the second option is selected, then the value of peak probability will be the natural log of the parameter's initial value. If the first option is selected, then the value of highest probability will be the average of the natural logs of a parameter's lower and upper bounds, these being specified in the PEST control file.

PEST2LHS next writes the following text to the screen.

```
Parameter probability distributions must now be provided. This must be done
on a group-by-group basis.

Select uniform/triangular/normal distributions as [u/t/n].

Note that triangular is not allowed if some group members are log-
transformed.
```

Then, for each parameter group featured in the PEST control file, PEST2LHS asks

```
Enter distribution for parameter group "param_group" [u/t/n]:
```

Enter "u", "t" or "n" as appropriate. PEST2LHS constructs parameter probability distributions in the following way.

- If a uniform distribution is requested, the distribution extends between the parameter's lower and upper bounds as declared in the PEST control file. It the parameter is log-transformed, PEST2LHS informs LHS that the parameter's distribution is log-uniform. Otherwise it is uniform.
- If a parameter group is endowed with a triangular distribution PEST2LHS first checks whether any member of that group is log-transformed. If any parameter within the group is, in fact, log-transformed, then PEST2LHS informs the user of this and repeats the above prompt. If no group member is log-transformed, then each parameter within the group is endowed with a triangular probability distribution whose bounding zero probability points are the lower and upper parameter bounds (as specified in the PEST control file), and whose peak probability point is either the midpoint of the bounds interval, or the parameter's initial value, as already discussed.
- If the "normal" option is selected in response to the above prompt, then each parameter within the group is awarded either a normal distribution (if it is untransformed) or a log-normal distribution (if the parameter is log-transformed in the PEST control file). If a parameter is untransformed and if in response to a previous prompt the parameter value of maximum probability is selected as the midpoint of the lower-to-upper bounds interval, then the standard deviation of the normal distribution is calculated as a sixth of the distance between the parameter's upper and lower bounds; the bounds are thus decreed to span a probability range of 99.7%. However these are made to span 100% of its probability range by declaring to LHS that the normal distribution is bounded, with the bounds supplied to LHS coinciding with parameter bounds provided in the PEST control file.
- If the normal option is selected and a parameter is log-transformed, the abscissa corresponding to the maximum value of the "underlying normal distribution" required by LHS is calculated as the midpoint of the distance between the natural logs of the parameter's bounds (i.e. the average of its natural logged lower and upper bounds), or is calculated as the natural log of the parameter's initial value; see above. If, in response to the pertinent previous prompt, the parameter value of maximum probability is selected as the midpoint of the lower-to-upper parameter bounds interval, then the standard deviation of the "underlying normal distribution" is calculated as a sixth of the difference between the natural logs of the parameter's lower and upper bounds. A bounded log-normal distribution is then specified to LHS, with bounds coinciding with the natural logs of the parameter's lower and upper bounds.
- If, in either of the transformed or untransformed normal distribution cases, the parameter value of maximum likelihood is specified as the parameter's initial value (or its log), the standard deviation ascribed to the (log)normal distribution associated with that parameter is equated to a third of the distance between the (log of) the parameter's initial value and (the log of) its lower or upper bound, whichever is further away. The bounded (log)normal distribution whose specifications are supplied to LHS may thus be asymmetric.

Uncertainty File

We begin this subsection by repeating a previous PEST2LHS prompt.

```
There are two options for specifying parameter probability distributions:
   For (log)normal distributions and an uncertainty file           - enter 1
   For user-specified distributions with bounds-derived descriptors - enter 2
Enter your choice:
```

As stated above, the PEST2LHS internal algorithm is such that use of a parameter uncertainty file implies selection of a (log)normal distribution for all parameters. The choice between a normal and a log-normal distribution is made on the basis of the transformation status of each parameter as specified in the PEST control file. On all occasions the (log)normal distribution is bounded, the bounds being those supplied as upper and lower parameter bounds in the PEST control file.

Following the above prompt, PEST2LHS's next prompt is (regardless of which of the above options is selected)

```
For maximum value of non-uniform distributions:-
   If halfway between (log) bounds                    - enter 1
   If parameter initial value from PEST control file  - enter 2
Enter your choice:
```

The mean of the (log)normal distribution as it applies to each parameter is calculated on the basis of the response to this prompt. If the first of the above options is selected, and if a parameter is untransformed, PEST2LHS calculates the mean of the parameter's distribution as the mean of its upper and lower parameter bounds. If a parameter is log-transformed, the mean of the LHS "underlying normal distribution" (i.e. the distribution pertaining to the natural log of the parameter) is calculated as the mean of the natural logs of the parameter's upper and lower bounds.

If the second of the above options is chosen, the mean assigned to the parameter's normal distribution is the parameter's initial value if the parameter is untransformed. Alternatively, if the parameter is log-transformed, the mean of the "underlying normal distribution" is calculated as the natural log of the parameter's initial value as specified in the PEST control file.

PEST2LHS next asks

```
Enter name of parameter uncertainty file:
```

The format of this file is described section 2.5 of this manual. As therein described, each parameter can be assigned a standard deviation; alternatively, groups of parameters can be assigned a covariance matrix. Where a parameter is untransformed, these apply to native parameters. Where a parameter is log-transformed they apply to the logged (to base 10) parameters.

Where a parameter uncertainty file is provided, PEST2LHS uses standard deviations from that file to assign standard deviations to parameter probability distributions that it provides to LHS. Where a parameter is untransformed, its standard deviation is directly transferred to the LHS input file. Where a parameter is log-transformed, PEST2LHS transfers the standard deviation from the uncertainty file to the LHS input file as the standard deviation of that parameter's "underlying normal probability distribution". However in doing so, account is taken of the fact that the parameter uncertainty file assumes log transformation to base 10 of parameters whereas LHS uses natural logs. Note also that when a parameter features in a

covariance matrix cited in the parameter value file, its (log)standard deviation is calculated as the square root of its variance as cited in that file.

Where a covariance matrix is provided for groups of parameters, PEST2LHS evaluates correlations between individual parameter pairs so that these can be recorded on the LHS input file; these correlations can thereby be taken into account in generating Latin hypercube sample sets. Where parameters are log-transformed, PEST2LHS calculates correlations pertaining to native parameters from the information provided in a pertinent covariance matrix, notwithstanding the fact that this information pertains to logged (to base 10) parameters. This satisfies LHS input requirements with respect to the CORR keyword. Correlations supplied to LHS are calculated from native variances/covariances using the usual formula for calculation of correlation coefficients. That is

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_i^2 \sigma_j^2}} \tag{13.4.1}$$

Where necessary, variances and covariances of untransformed parameters are calculated from those of their log-transformed counterparts using the formula

$$\sigma_{ij} = e^{\left[(v_i + v_j) + (\delta_i + \delta_j)/2\right]} \left(e^{\delta_{ij}} - 1\right) \tag{13.4.2}$$

where v_i and v_j are means of the normal distribution pertaining to the logs of parameters i and j, and δ_i, δ_j and δ_{ij} are the log parameter covariance matrix elements pertaining to these parameters. Means are calculated in the manner described above (i.e. as logged parameter range midpoints, or as logs of initial parameter values, according to the user's selection).

In the unlikely event that a covariance matrix is supplied for a group of parameters of which some are log-transformed and some are not, PEST2LHS simply uses covariances from the composite group covariance matrix in order to compute correlation coefficients for parameter pairs. These correlation coefficients are then written directly to the LHS input file.

LHS Input Files

Regardless of the methodology selected for specification of parameter probabilities, PEST2LHS's final prompts are

```
Enter filename base for LHS files:
How many parameter sample sets must be generated?
Enter random number seed (a big integer):
```

Suppose that you respond to the first of the above responses with the name "*file*". PEST2LHS will then write an LHS input file named *file.dat*. In that file it will specify that the LHS sample data output file (this being associated with the LHS keyword LHSOUT) is named *file.lsp* and that the LHS-generated message file (this being associated with the LHS keyword LHSMSG) is named *file.msg*.

The other two prompts are self-explanatory.

14. Miscellaneous Utilities

14.1 Introduction

This chapter documents a number of utilities that do not fall into any of the categories defined by the previous chapters of this manual.

PARREDUCE suggests an optimal parameter reduction strategy in contexts where model run times are high and parameter parsimony is the only regularisation option. CALMAINTAIN derives a new parameter set where certain parameters in an existing parameter set have had their values altered; ideally, the new parameter set compensates for the altered values of the original parameter set in a way that maintains the model in a calibrated state. PESTLIN, and its partner program GENLIN, construct a linear model from a nonlinear model. This model can then be used as a fast-running surrogate for the original model where this is required for numerically intensive tasks such as optimisation. DERCOMB facilitates PEST usage where a model can calculate its own derivatives. It amalgamates derivatives computed by two different models into a single file which can then be read by PEST.

14.2 PARREDUCE

14.2.1 General

"PARREDUCE" stands for "parameter reduction". This utility can be used to ascertain the cost, in terms of model-to-measurement misfit, of a reduction in the number of parameters employed by a model. Better still, it can be used to compare the effectiveness of different parameter reduction strategies, where "effectiveness" is based on the assumption that those parameter reduction strategies are better which incur simplification-induced misfit the least.

Use of PARREDUCE assumes that parameter reduction can occur in one of three ways:

- Through removal of parameters from a PEST control file;
- Through fixing parameters within a PEST control file (which is the same as removing them); and/or
- Through tying certain parameters to other parameters, so that the parent parameters are estimated with their child parameters "riding on their backs". (This is effectively the same as parameter amalgamation.)

Reduction in the number of parameters employed by a model has the obvious advantage that the time required for PEST to calibrate the model is thereby reduced. However savings in computational burden come at a cost; in particular, when the number of parameters employed by a model is reduced, the objective function achieved through the calibration process may increase. Generally, the extent to which this occurs will only be known after PEST has been run to calibrate the simplified model. It is possible that PEST can achieve just as good a fit with a simplified parameter set as with a complex parameter set by adjusting simplified parameter values appropriately. However if the simplified parameter set does not span the calibration solution space of the complex model, model-to-measurement fit will be compromised.

The PARREDUCE utility can be used not just for optimisation of model simplification; it can also be used to examine how best to add parameters to a model. A modeller may commence the complexifying processing by adding many parameters to a model – more than he/she

ultimately wishes to add. He/she can then test different ways of amalgamating the supplementary parameters in ways that reflect expert knowledge of, for example, underlying geological variability, before deciding on an ultimate parameter complexification strategy that achieves most reduction in the objective function with the least number of additional parameters. Parameter simplification thus takes place as the second step of an overall parameter complexification process.

PARREDUCE provides information on the relative cost of different simplification strategies. It does this by providing an indication of the expected increase in the objective function incurred by use of fewer model parameters. It must be pointed out however, that the numbers which it calculates are indicative only. No account is taken of the contribution made by measurement noise to the objective function. Nor is account taken of simplification that has already been undertaken in building the original, pre-simplified model. Furthermore calculations undertaken by PARREDUCE rely on a linearity assumption. Sensitivities on which linear analysis is based are calculated using the parameter set currently employed by the model.

14.2.2 Theory

The theory presented in this subsection summarises that presented in section 7.5 of Doherty (2015).

Let the action of a complex model on its parameters be described by the matrix **Z**. Use of a matrix as a substitute for the model implies that the relationship between model parameters and model outputs is linear. Let **k** be the parameter set employed by a "complex" model. Under calibration conditions

$$\mathbf{h} = \mathbf{Z}\mathbf{k} + \boldsymbol{\varepsilon} \tag{14.2.1}$$

where

h is the vector of measurements comprising the calibration dataset; and

ε is the vector of noise associated with elements of this dataset.

Measurement noise is now assumed to be zero so that attention can be focussed on the outcomes of parameter simplification. Let the vector **p** comprise a simplified parameter set. Let model outputs **o** corresponding to measurements **h** be computed by the simplified model using the matrix **X**. Thus

$$\mathbf{o} = \mathbf{X}\mathbf{p} \tag{14.2.2}$$

Note that the **Z** and **X** matrices used by PARREDUCE are, in fact, weighted sensitivity matrices; observation weights are provided through a PEST control file.

Let the relationship between **Z** and **X** be described by a matrix relationship of the form

$$\mathbf{X} = \mathbf{Z}\mathbf{L} \tag{14.2.3}$$

Often **L** is a kind of "selection matrix". Thus, for example, if the parameter scheme employed by the simplified model is comprised of zones of piecewise constancy, and the complex model allows for the existence of a greater number of these zones defined through subdivision of the broad scale zonation scheme embodied in **X**, then the value assigned to each element of **k** is equal to that of the element of **p** that characterizes the broad scale zone in which the **k** element lies. The **L** matrix is thus comprised of appropriately-positioned 1's and 0's.

Use of PARREDUCE does not require that the matrix **L** be explicitly defined. However its use assumes that two PEST control files exist, one embodying a complex parameter set and the other containing a simpler parameter set defined in ways discussed above (i.e. parameter omission, tying and fixing). For these types of parameter simplification, **X** is easily calculated from **Z**. Calculations undertaken by PARREDUCE are in fact exactly the same as those undertaken by JCO2JCO when it is presented with two PEST control files, one of which is a simplification of the other. In both cases the **Z** matrix occupies the JCO file corresponding to the complex PEST control file.

Let the covariance matrix of the prior probability distribution of **k** be denoted as C(**k**). This therefore describes the "innate variability" of parameters **k**, this being the encapsulation of expert knowledge applied to **k**. PARREDUCE can be asked to assume that C(**k**) is the identity matrix **I**. Alternatively it can be asked to assume that C(**k**) is a diagonal matrix in which each parameter is statistically independent of all other parameters, and in which the standard deviation of each parameter is calculated to be a quarter of the difference between upper and lower parameter bounds provided in the PEST control file. If a parameter is log-transformed the covariance matrix applies to the log of each parameter; parameter bounds are therefore logged before differencing in this case. The diagonal elements of C(**k**) are parameter variances, these being squares of parameter standard deviations.

Residuals incurred by parameter simplification are expressed as

$$\mathbf{r} = \mathbf{Z}\mathbf{k} - \mathbf{X}\mathbf{p} \tag{14.2.4}$$

If a model is not calibrated at all (and **Xp** is effectively **0**), then the covariance matrix of residuals, denoted here as $C_u(\mathbf{r})$, is readily calculated as

$$C_u(\mathbf{r}) = \mathbf{Z}C(\mathbf{k})\mathbf{Z}^t \tag{14.2.5}$$

In this formula, "u" signifies "uncalibrated". Now suppose that the model **Z** is calibrated. Let it be assumed that this is done using truncated singular value decomposition; note that if the inverse problem is not ill-posed then this is the same as using the Gauss-Marquardt-Levenberg methodology of model calibration. A parameter set $\underline{\mathbf{k}}$ is thereby computed as

$$\underline{\mathbf{k}} = \mathbf{V}_{z1}\mathbf{S}^{-1}_{z1}\mathbf{U}^t_{z1}\mathbf{h} \tag{14.2.6}$$

Here the "z" subscript indicates that singular value decomposition is performed on the **Z** matrix. Meanwhile the "1" subscript indicates partitioning of pertinent matrices, and restriction of their contents to the solution subspace of parameter space, i.e. the subspace that is retained after singular value truncation. See Doherty (2015) for details. Post-calibration residuals are then calculated as

$$\mathbf{r}_z = \mathbf{Z}\mathbf{k} - \mathbf{Z}\underline{\mathbf{k}} \tag{14.2.7}$$

Non-zero-valued residuals calculated through equation 14.2.7 can occur as an outcome of the existence of small non-zero singular values to the right of the singular value truncation point. These comprise the $\mathbf{S}_{z2}$ matrix not represented in equation 14.2.6. As Doherty (2015) explains, retainment of these singular values in the inversion process would result in amplification of measurement noise; hence the subspace of parameter space that is associated with these residuals is relegated to the null space to forestall this occurrence.

With a little matrix manipulation, equation 14.2.7 becomes

$$\mathbf{r}_z = \mathbf{U}_{z2}\mathbf{S}_{z2}\mathbf{V}^t_{z2}\mathbf{k} \tag{14.2.8}$$

The covariance matrix of **Z**-calibration residuals can then be calculated as

$$C_z(\mathbf{r}) = \mathbf{U}_{z2}\mathbf{S}_{z2}\mathbf{V}^t_{z2}C(\mathbf{k})\mathbf{V}_{z2}\mathbf{S}_{z2}\mathbf{U}^t_{z2} \quad (14.2.9)$$

If

$$C_z(\mathbf{k}) = \mathbf{I} \quad (14.2.10)$$

this becomes

$$C_z(\mathbf{r}) = \mathbf{U}_{z2}\mathbf{S}^2_{z2}\mathbf{U}^t_{z2} \quad (14.2.11)$$

The diagonal elements of $C_z(\mathbf{r})$ provide the variances of **Z**-calibration residuals corresponding to the observations **h**. The square roots of these variances provide their post-calibration standard deviations.

It is important to note that calculation of $C_z(\mathbf{r})$ in the manner described above is based on two assumptions. These are as follows.

- Measurement noise is zero;
- The model **Z** is "perfect" in the sense that it is able to fit the data perfectly; that is, "structural noise" makes no contribution to **r**.

$C_z(\mathbf{r})$ can thus be very small. For the types of analyses that PARREDUCE is asked to perform this is fine. Its purpose is to provide a means of comparison of residuals that emerge from calibration of the **Z** model with those that emerge from calibration of various simplified **X** models.

Where an **X** model is calibrated in place of the **Z** model, equation 14.2.7 is replaced by equation 14.2.4; that is

$$\mathbf{r}_x = \mathbf{Z}\mathbf{k} - \mathbf{X}\underline{\mathbf{p}} \quad (14.2.12)$$

where $\underline{\mathbf{p}}$ is calculated as

$$\underline{\mathbf{p}} = \mathbf{V}_{x1}\mathbf{S}^{-1}_{x1}\mathbf{U}^t_{x1}\mathbf{h} \quad (14.2.13)$$

Here the subscript "x" indicates that singular value decomposition is undertaken on **X** rather than on **Z**. Substitution of equation 14.2.13 and equation 14.2.1 (with **ε** assumed to be **0**) into equation 14.2.12 yields

$$\mathbf{r}_x = \mathbf{Z}\mathbf{k} - \mathbf{X}\mathbf{V}_{x1}\mathbf{S}^{-1}_{x1}\mathbf{U}^t_{x1}\mathbf{Z}\mathbf{k} \quad (14.2.14)$$

which (after a little matrix manipulation) becomes

$$\mathbf{r}_x = (\mathbf{I} - \mathbf{U}_{x1}\mathbf{U}^t_{x1})\mathbf{Z}\mathbf{k} = \mathbf{U}_{x2}\mathbf{U}^t_{x2}\mathbf{Z}\mathbf{k} \quad (14.2.15)$$

The covariance matrix of residuals emerging from calibration of a simplified model is then calculated as

$$C_x(\mathbf{r}) = \mathbf{U}_{x2}\mathbf{U}^t_{x2}\mathbf{Z}C(\mathbf{k})\mathbf{Z}^t\mathbf{U}_{x2}\mathbf{U}^t_{x2} \quad (14.2.16)$$

which, if $C(\mathbf{k})$ is equal to **I**, becomes

$$C_x(\mathbf{r}) = \mathbf{U}_{x2}\mathbf{U}^t_{x2}\mathbf{Z}\mathbf{Z}^t\mathbf{U}_{x2}\mathbf{U}^t_{x2} \quad (14.2.17)$$

Once again, the diagonal elements of $C_x(\mathbf{r})$ represent the variances of individual elements of **r**; the square roots of these represent their standard deviations.

Different simplification strategies can be compared through comparing $C_x(\mathbf{r})$ matrices calculated using equation 14.2.16 for different **X** matrices. A simple summary statistic is the sum of elements along the diagonal of $C_x(\mathbf{r})$. This is denoted as Φ_x as it is an objective function. Thus

$$\Phi_x = \mathrm{tr}(C_x(\mathbf{r})) \qquad (14.2.18)$$

where tr() is the matrix trace operator. Φ_x can be thought of as the contribution made to the overall calibration objective function by parameter simplification. If this number is low (or even zero), it indicates that the simplified model retains enough adjustable parameters to guarantee a fit with field data that is very little worse than that which would prevail through calibration of the original **Z** model. This does not mean that simplification is therefore "ideal". It is possible that the simplified parameters play roles that compensate for model defects in achieving this level of fit. Nevertheless it does indicate that enough parameters exist for a good fit between model outputs and field data to be attained, and that these parameters therefore span the solution space of the **Z** model.

In the subsurface reservoir modelling context, different simplification strategies (embodied in different **X** matrices) may result from alternative representations of the presence, locations and geometries of zones of piecewise constancy reflecting rock types of contrasting, but relatively homogeneous, hydraulic properties. Lower values for Φ_x arising from some of these tested dispositions may indicate better representation of subsurface features in the model domain.

14.2.3 Using PARREDUCE

PARREDUCE is run using the command

```
parreduce pestfile1 pestfile2 eigthresh outfile [/b]
```

where

pestfile1	is a PEST control file for which a JCO file exists,
pestfile2	is a PEST control file in which some parameters are tied and/or fixed,
eigthresh	is an eigenvalue truncation threshold,
outfile	is the name of the PARREDUCE output file, and
/b	activates optional parameter bounds scaling.

Use of PARREDUCE assumes that you have constructed a PEST control file embodying complex parameterization, and that you have run PEST to obtain a corresponding JCO file (for example by setting NOPTMAX to -2 in the PEST control file and then running PEST). You must then construct another PEST control file in which all observations are the same as in the original PEST control file, but in which some parameters are tied to other parameters, some parameters are fixed, and/or some parameters are omitted altogether. This constitutes the second PEST control file which must be cited in the PARREDUCE command line above. No JCO file need accompany this second PEST control file.

The EIGTHRESH variable must be set between 0.0 and 1.0. This is the ratio of squared singular values of the weighted **Z** matrix (i.e. $\mathbf{Q}^{1/2}\mathbf{Z}$) which determines where singular value truncation occurs. (The ratio is taken of each squared singular value to the largest squared singular value.) As is explained by Doherty (2015), this value should be set at a level that minimizes post-calibration predictive error variance. The latter falls as EIGTHRESH is reduced below 1.0, for the inclusion of extra singular values expands the solution space (spanned by the vectors which comprise the columns of $\mathbf{V}_1$) and shrinks the null space (spanned by the vectors which comprise the columns of $\mathbf{V}_2$). Beyond a certain value of this truncation threshold however, post-calibration predictive error variance starts to rise due to amplification of measurement noise. EIGTHRESH should never be set lower than about 5E-7, as this is the level at which amplification of numerical noise seriously degrades predictive performance of a calibrated model. (EIGTHRESH is the same variable that

appears in the "singular value decomposition" section of a PEST control file.)

If the optional "/b" switch appears in the PARREDUCE command line, then PARREDUCE estimates C(**k**) by assuming that the differences between parameter upper and lower bounds define four standard deviations for each parameter, and that no correlation exists between parameters. Alternatively, if the "/b" switch is omitted, C(**k**) is assumed to be the identity matrix **I**. This may be suitable for comparing different simplification strategies where all parameters are log-transformed, and where all sensitivities with respect to these parameters are thereby effectively normalized by their values.

A PARREDUCE output file for a small problem is provided below. The first column provides the names of observations used in the inversion process. The second column provides the square roots of the diagonal elements of $C_u(\mathbf{r})$ calculated using equation 14.2.5. The third column provides the square roots of the diagonal elements of $C_z(\mathbf{r})$ calculated using equation 14.2.9, while the last column contains the square roots of the diagonal elements of $C_x(\mathbf{r})$ calculated using equation 14.2.16. The traces of the pertinent C(**r**) matrices are shown at the bottom of each column.

```
Residual standard deviations (no measurement noise assumed)

Observation           No_calibration         Calibration_with_Z     Calibration_with_X
  ar1                    0.7916715              5.1651679E-02          0.1171594
  ar2                    0.7531351              4.6104210E-02          0.1354490
  ar3                    0.7147344              4.6648685E-02          5.9892814E-02
  ar4                    0.6894756              2.4442489E-02          6.2925528E-02
  ar5                    0.6692790              4.8524822E-02          7.2914701E-02
  ar6                    0.6568247              4.2261614E-02          3.2690919E-02
  ar7                    0.6470027              3.2050366E-02          3.4495961E-02
  ar8                    0.6399243              5.3340646E-02          4.8791286E-02
  ar9                    0.6358643              3.1437273E-02          9.4707001E-02
  ar10                   0.6321911              4.1518229E-02          0.1373004
  ar11                   0.6304041              5.1209897E-02          9.2412584E-02
  ar12                   0.6291723              2.0590739E-02          6.4143446E-02
  ar13                   0.6287475              5.1769681E-02          0.1718363
  ar14                   0.6325852              3.4882597E-02          7.6345205E-02
  ar15                   0.6576022              2.2096805E-02          8.9699859E-02
  ar16                   0.7414070              4.2101009E-02          0.1366652
  ar17                   0.8908967              2.1390807E-02          6.5164786E-02
  ar18                    1.040221              7.5365578E-03          3.0688386E-02
  ar19                    1.140819              3.0784107E-02          0.1014441

  sum_of_squares          6.869226              5.8772489E-05          2.5599931E-03
```

Figure 14.1 A PARREDUCE output file.

14.3 CALMAINTAIN

14.3.1 General

Suppose that you have calibrated a model, and that you have built a PEST control file in which initial parameter values are calibrated parameter values. Suppose also that you have set NOPTMAX to -2 in this file and run PEST in order to obtain a JCO file in which finite-difference derivatives are calculated based on the calibrated parameter set.

Suppose now that you wish to alter the values of a number of parameters from those in this PEST control file, and would like to adjust the values of other parameters to compensate for this so that the model remains calibrated. The purpose of CALMAINTAIN is to perform these compensatory parameter value alterations. The success achieved by CALMAINTAIN in computing a set of parameters which maintain the model in a calibrated state despite user-adjustment of the values of other parameters will depend on a number of factors. These include the following.

- The linearity of model outputs with respect to parameters (as is explained below, CALMAINTAIN uses linear theory to compute a new set of parameters);
- The degree to which the model was de-calibrated by adjusting the values of some of its parameters in the first place;
- The extent to which the existence or otherwise of a null space makes parameter compensation possible.

14.3.2 Theory

Let the action of a (linearized) model **Z** on its parameters **k** under calibration conditions be described by the equation

$$\mathbf{h} = \mathbf{Z}\mathbf{k} + \boldsymbol{\varepsilon} \tag{14.3.1}$$

where $\boldsymbol{\varepsilon}$ is a vector of measurement noise. If the parameter set **k** is partitioned into two sets of parameters $\mathbf{k}_1$ and $\mathbf{k}_2$, and if measurement noise is neglected, this equation becomes

$$\mathbf{h} = \begin{bmatrix}\mathbf{Z}_1 & \mathbf{Z}_2\end{bmatrix}\begin{bmatrix}\mathbf{k}_1 \\ \mathbf{k}_2\end{bmatrix} = \mathbf{Z}_1\mathbf{k}_1 + \mathbf{Z}_2\mathbf{k}_2 \tag{14.3.2}$$

If parameters $\mathbf{k}_2$ are altered by $\delta\mathbf{k}_2$, the model will remain in a calibrated state if a set of alterations $\delta\mathbf{k}_1$ to $\mathbf{k}_1$ can be found such that

$$\mathbf{Z}_1\delta\mathbf{k}_1 = -\mathbf{Z}_2\delta\mathbf{k}_2 \tag{14.3.3}$$

This can only occur if

$$\begin{bmatrix}\mathbf{Z}_1 & \mathbf{Z}_2\end{bmatrix}\begin{bmatrix}\delta\mathbf{k}_1 \\ \delta\mathbf{k}_2\end{bmatrix} = \mathbf{0} \tag{14.3.4}$$

That is, if the matrix **Z** has a null space. If it does not have a null space, or if equation 14.3.3 cannot be solved because one or a number of elements of $\delta\mathbf{k}_2$ lies entirely within the solution space, then $\delta\mathbf{k}_1$ should be chosen such that the right side of equation 14.3.4 is minimized. If this is done, then the model maintains respect for the calibration dataset as much as possible following user-alterations to $\mathbf{k}_2$.

CALMAINTAIN solves equation 14.3.3 in the exact sense if it can, and in the least squares sense if it cannot. Thus it finds a $\delta\mathbf{k}_1$ to compensate for a user-imposed $\delta\mathbf{k}_2$.

Note that the starting equation for CALMAINTAIN's calculations is actually not equation 14.3.1 at all; instead it uses the following equation, where **Q** is the weight matrix.

$$\mathbf{Q}^{1/2}\mathbf{h} = \mathbf{Q}^{1/2}\mathbf{Z}\mathbf{k} + \mathbf{Q}^{1/2}\boldsymbol{\varepsilon} \tag{14.3.5}$$

14.3.3 Using CALMAINTAIN

CALMAINTAIN prompts the user for its input data as this is more convenient than supplying these data on the command line. Its first prompt is

```
Enter name of PEST control file:
```

CALMAINTAIN opens the PEST control file. At the same time it ensures that a corresponding JCO file exists. (The contents of this file become the **Z** matrix of the above equations). Next it asks

```
Enter name of parameter adjustment file:
```

An example of a parameter adjustment file follows.

```
# Example of a parameter adjustment file
k_ppt30    0.8
k_ppt31    2.6

k_ppt32    0.8

# Here is another comment
k_ppt33    1.0
k_ppt34    0.8
k_ppt35    0.8
k_ppt36    0.5
k_ppt37    1.8
```

Figure 14.2 Example of a parameter adjustment file.

Each line of a parameter adjustment file must have two entries. The first entry is the name of a parameter. The second entry is the new value of that parameter. Blank lines are ignored. Comment lines (which begin with the "#" character) are also ignored.

CALMAINTAIN will object if you attempt to adjust the value of a fixed or tied parameter. However it is acceptable to adjust the value of a parent parameter to which other parameters are tied; the child parameters are automatically adjusted if this is the case.

CALMAINTAIN next asks for the name of the parameter value file that it must write. The prompt is

```
Enter name for output parameter value file:
```

It then asks the following three questions.

```
Use SVD or LSQR to solve equation? [l/s]:
Truncate at how many singular values:
Adopt what fraction of upgrade vector:
```

As is apparent from the first of the above questions, you have the choice of using either LSQR or singular value decomposition to solve equation 14.3.3. LSQR is faster than singular value decomposition where parameter numbers are high, but can sometimes be less precise. The second of the above prompts allows you to limit the propensity for numerical noise to contaminate the solution of equation 14.3.3. The best response to this question varies with context. Based on limited experience at the time of writing, it is suggested that a value equal to the number of adjusted parameters featured in the parameter adjustment file may work well. "Damping" of parameter compensation can also be used to limit contamination of the solution to equation 14.3.3 by numerical noise; therefore a value of between 0.0 and 1.0 is a suitable answer to the third of the above questions (normally closer to 1.0 than 0.0.). Some trial and error may be necessary to find the best response to the second and third of the above questions in any particular modelling context.

Once all of the above questions have been answered, CALMAINTAIN performs the calculations indicated above, and then writes the requested parameter value file. The values assigned to parameters in this file are the user-adjusted values (for those parameters which are cited in the parameter adjustment file), together with values for other parameters which are altered in such a way as to maintain the model in as much of a calibrated state as possible. If desired, the PARREP utility can be used to insert parameter values recorded in this file into a PEST control file. If NOPTMAX is set to zero in this file, PEST can be used to run the model and calculate an objective function. In this way adherence to calibration constraints can be tested.

14.4 GENLIN

14.4.1 General

"GENLIN" stands for "general linear model". The main purpose of GENLIN is to complement PESTLIN functionality (see below) through which a nonlinear model can be replaced by its linearised equivalent for purposes such as predictive uncertainty analysis and other kinds of optimisation. However GENLIN can also be employed on a stand-alone basis without the use of PEST at all.

GENLIN calculates its outputs **h** from its inputs **k** using the equation

$$\mathbf{h} = \mathbf{h}_0 + \mathbf{M}(\mathbf{k} - \mathbf{k}_0) \qquad (14.4.1)$$

where **M** is a matrix, and **h**, $\mathbf{h}_0$, **k** and $\mathbf{k}_0$ are vectors. For the sake of conformity with PEST terminology (and in conformity with use of GENLIN as an adjunct to PESTLIN), the **k** vector will normally contain parameter values while the **h** vector will contain the model-generated counterparts to field measurements, or simply "observations" in PEST parlance. $\mathbf{k}_0$ and $\mathbf{h}_0$ 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 **h** and as many columns as there are elements of **k**. GENLIN allows parameters to be log-transformed. In this case, columns 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

$$\mathbf{h} = \mathbf{h}_0 + \mathbf{M}[\log(\mathbf{k}) - \log(\mathbf{k}_0)] \qquad (14.4.2)$$

where the log is taken to base 10.

14.4.2 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
```

Figure 14.3 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 Section

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 Section

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 four 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 (**k** in the above equations), while the fourth is its "offset value" ($\mathbf{k}_0$ 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 individual parameters. Where a parameter is designated as log-transformed, GENLIN takes care of logarithmic

transformation itself.

Observations Section

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 ($\mathbf{h}_0$ in the above equations).

Sensitivities Section

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. a JCO file). If not, ASCII storage in matrix file format is assumed; the format of this kind of file is discussed in section 2.4 of this manual. Note that the JCO2MAT utility can be used to convert a JCO file to a matrix file.

It is important that the sensitivity file matches the specifications for the linear model 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 all observations to that parameter 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.

14.4.3 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). Rows of this matrix retain the same names as observations 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 can also check 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 it is not necessary that extensive, and possibly time-consuming, checks of all aspects of input data integrity be checked on every occasion that GENLIN is run.

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 derivatives, thereby saving PEST the need to carry out many GENLIN runs in the course of its execution. See chapter 12 of part I of this manual for documentation of PEST's external derivatives functionality. Note that, in accordance with PEST's requirements, derivatives written to this file pertain to native parameters rather than to log-transformed parameters; hence where parameters are log-transformed the entries of this file will differ from those of the GENLIN input sensitivity file.

14.5 PESTLIN

14.5.1 General

Use of PESTLIN assumes that a complete PEST input dataset exists for parameter estimation (regularised or otherwise), or predictive uncertainty analysis. It also assumes that a Jacobian matrix file 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 PESTLIN writes a GENLIN (see above) input dataset, encapsulating a linearised form of the model. It also writes a new PEST control file so that (regularised) inversion or predictive uncertainty 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 to build a linearised equivalent of a PEST input dataset (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.
4. Run PESTLIN.

Alternatively, PESTLIN can be run at the end of an entire 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). Meanwhile the run record file will record optimised parameter values, together with model-generated counterparts to measurements comprising the calibration dataset calculated on the basis of these optimised parameter values (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 and then running PEST in the manner described above).

14.5.2 Using PESTLIN

PESTLIN is run using the command

```
pestlin pestincase pestoutcase linbasename [/d]
```

where

pestincase	is the filename base of a PEST input dataset,
pestoutcase	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 *pestincase* supplied on the PESTLIN command line to obtain specifications of the current parameter estimation problem.
2. It reads the corresponding Jacobian matrix file *pestincase.jco* in order to obtain sensitivities of model outputs to all adjustable parameters.
3. It reads the corresponding run record file *pestincase.rec* in order to obtain the values of adjustable parameters on which 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. Through this file GENLIN is instructed to write an output file named *linbasename.out*, and to read sensitivities of adjustable parameters from the file *pestincase.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 *linbasename.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 *pestoutcase.pst* which formulates the same parameter estimation problem as that formulated by the original PEST control file *pestincase.pst*, but this time on the basis of a linearised model run using GENLIN.

PEST can then be run on the basis of *pestoutcase.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, parameters recorded in *pestoutcase.par* parameter value files lack these parameters. Fortunately, PARREP can still be employed to insert parameter values found in *pestoutcase.par* parameter value files into the original PEST input dataset encapsulated in *pestincase.pst* to create a new PEST control file. 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 a parameter value file does not affect their ability to be included in a new PEST control file written by PARREP.

14.6 DERCOMB1

14.6.1 General

"DERCOMB" stands for "derivatives combination". The "1" suffix was included in the name of this utility in anticipation of the fact that it may not be the last utility written to perform a similar role.

DERCOMB1 was written for use in calibration contexts where a model provides its own derivatives. As explained in chapter 12 of part I of this 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 files.

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 program. 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 derivative matrix elements which have not been determined.

The second derivatives file must be in PEST matrix file format (see section 2.4 of this manual for specifications of this file type). A derivatives file in matrix format is written, for example, by the ZONE2VAR1 Groundwater Data Utility program. 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 in this file as they are represented in the PEST control file which defines the current inverse problem.)

DERCOMB1 then reads the matrix file. Each 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.

14.6.2 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 derivatives 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 replaces derivatives read from the existing external derivatives file with those read from 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. (Note that DERCOMB1 does not read or write binary derivatives files.)

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 component that writes the external derivatives file as well as the model component that writes the matrix file of derivatives. Where there are many model executables and each of them calculate some of the derivatives required by PEST, DERCOMB1 may be run multiple times as part of the one model.

15. Matrix Manipulation Programs

15.1 Introduction

This chapter documents a series of matrix manipulation programs.

Many of the PEST utility programs documented in this manual read and/or write files which contain matrices. The format used for these files is documented in section 2.4 of this manual. Often-used matrices include the following.

- The Jacobian matrix recorded by PEST;
- The $\mathbf{J}^t\mathbf{QJ}$, or so-called "normal matrix", used by PEST as a basis for solution of an over-determined inverse problem;
- Observation covariance matrices supplied to PEST instead of observation weights;
- Parameter and measurement noise covariance matrices used in error and uncertainty analysis; and
- Predictive sensitivity vectors (recall that a vector is simply a one-dimensional matrix) used in predictive error and uncertainty analysis.

A number of programs documented in chapter 5 of this manual provide translation capabilities between JCO and matrix files. They are

- JCO2MAT, which rewrites a Jacobian matrix housed in a JCO file in PEST matrix file format;
- MAT2JCO, which rewrites a Jacobian matrix stored in PEST matrix file format as a JCO file;
- JROW2VEC, which extracts a row from a JCO file, recording the contents of that row as a vector (i.e. as a matrix with only one column);
- JROW2MAT, which extracts a row from a JCO file, recording the contents of that row as a matrix (with only one row).

In contrast to other chapters of this manual, programs are described in alphabetical order in the present chapter.

15.2 COV2COR

COV2COR calculates a correlation coefficient matrix from a covariance matrix. It reads the former from a file which observes PEST matrix file format and writes the latter to a file which observes the same format. The matrix in the former file must satisfy the following requirements.

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.

COV2COR is run using the command

```
cov2cor covfile corfile
```

where

covfile is the name of the matrix file holding the covariance matrix, and
corfile is the name of the file to which the corresponding correlation coefficient

matrix is written.

15.3 COVCOND

Suppose that an arbitrary vector **x** is partitioned into two separate parts $\mathbf{x}_1$ and $\mathbf{x}_2$. That is

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \tag{15.3.1}$$

Let C(**x**), the covariance matrix of **x**, be correspondingly partitioned as

$$\mathrm{C}(\mathbf{x}) = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{15.3.2}$$

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

$$\mathbf{C}'_{11} = \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}^{-1}{}_{22}\mathbf{C}_{21} \tag{15.3.3}$$

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 pertaining to the variables whose values are not known (i.e. $\mathbf{C}'_{11}$ in the above explanation).

As is explained in section 2.4 of this manual, a PEST-compatible matrix file includes a list of row and column names. For a covariance matrix file, row and column names are the same; presumably, these are 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.

15.4 MAT2CSV

MAT2CSV records a matrix in a CSV file.

MAT2CSV is run using the command

```
mat2csv matfile csvfile
```

where

matfile	is the name of a matrix file, and
csvfile	is the name of a new CSV file whose task it is for MAT2JCO to write.

Matrix column names are recorded along the top row of the new CSV file. Matrix row names comprise the first column of the CSV file.

15.5 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 JCO file.

MAT2JCO is run using the command

```
mat2jco matfile jcofile
```

where

matfile	is the name of a matrix file, and
jcofile	is the name of a new JCO file whose task it is for MAT2JCO to write.

15.6 MAT2SRF

MAT2SRF rewrites a matrix in SURFER grid file format. This can be a particularly useful device for viewing a resolution matrix (see utilities such as RESPROC and RESWRIT described elsewhere in this manual). 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 an inability of the calibration process to capture parameterisation detail, this arising from information deficits in the calibration dataset.

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 SURFER's "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.

15.7 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 transfers those provided in *matfile1* to the summation matrix recorded in file *matoutfile*. However it warns the user of the name-incompatibility between the matrices contained in *matfile1* and *matfile2*. MATADD does not therefore re-order the rows and columns of one matrix in order to ensure correspondence with that of the other 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.

15.8 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 $m \times n$ matrix. A new $m \times ncol$ 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 file in reverse order.

15.9 MATDIAG

MATDIAG extracts the diagonal of a matrix, writing it as a vector (i.e. as a one-column matrix). However certain conditions must be met for MATDIAG to do its job. These are as follows.

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 matfile matoutfile
```

where

matfile is the name of a file holding a square matrix, and
matoutfile is the name of the matrix file to which extracted diagonal elements of the matrix are written.

MATDIAG provides the rows of the one-column matrix which it records to *matoutfile* with the same names as the rows of original matrix. The single column of the new matrix is given the name “col1”.

15.10 MATDIFF

MATDIFF is identical to MATADD except for the fact that it undertakes matrix differencing instead of matrix addition.

15.11 MATINVP

MATINVP calculates the inverse of a positive definite matrix. It is run using the command

`matinvp matfile matoutfile`

where

matfile contains the matrix to be inverted, and
matoutfile contains the matrix to which the inverse is written.

If the matrix contained in *matfile* is not positive definite, MATINVP ceases execution with an appropriate error message.

15.12 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 the single matrix **C** constructed as

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \tag{15.10.1}$$

MATJOINC is run using the command

`matjoinc matfile1 matfile2 matoutfile`

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
matoutfile 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 (see section 2.4) for the sake of storage efficiency.

15.13 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

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix} \tag{15.11.1}$$

MATJOIND is run using the command

```
matjoind matfile1 matfile2 matoutfile
```

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
matoutfile 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 (see section 2.4) for the sake of storage efficiency.

15.14 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. Suppose that the two existing matrices are named **A** and **B**. MATJOINR combines these matrices into a single matrix **C** of the form

$$\mathbf{C} = [\mathbf{A} \quad \mathbf{B}] \tag{15.12.1}$$

MATJOINR is run using the command

```
matjoinr matfile1 matfile2 matoutfile
```

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
matoutfile 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 (see section 2.4) for the sake of storage efficiency.

15.15 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 a "reordering matrix", and
matoutfile	is the name of a new matrix file which will hold the reordered matrix.

MATORDER commences execution by reading the matrix contained in file *matfile1*. It then reads file *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. (Ordering is by row and column name.) 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 matrix elements can be stored; see section 2.4. For example each row of a 1000-column null matrix **0** can be represented using the string "1000*0.0".

15.16 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 of **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 the 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.

15.17 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 matoutfile
```

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
matoutfile is the output matrix file.

The following should be noted.

1. MATQUAD requires an input vector **y**. However this vector is actually read as a $n \times 1$ 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 the 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 from the columns of **M**.

15.18 MATROW

MATROW extracts a row of a matrix. It then re-writes that row as a "row matrix" to a matrix file.

An interesting use of MATROW is the extraction of a row of a resolution matrix. The MATTRANS utility can then be used to write row entries in the vertical direction rather than in 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. Gridding and contouring of this data then allows graphical viewing of the averaging process that attends the estimation of spatial model parameters where their estimation takes place through an ill-posed inverse problem.

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.

15.19 MATSMUL

MATSMUL multiplies a matrix by a scalar. It is run using the command

```
matsmul matfile number matoutfile
```

where

matinfile is a file containing a matrix,
number is the scalar multiplier, and
matoutfile is the file to which the new matrix is written.

15.20 MATSPEC

MATSPEC lists some useful matrix specifications to a nominated text file, namely:

1. the number of rows and columns in the matrix;
2. the row/column numbers/names of highest and lowest matrix elements;
3. the row/column numbers/names of highest and lowest absolute matrix elements;

4. the row/column numbers/names of highest and lowest diagonal elements;
5. the 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.

15.21 MATSVD

MATSVD undertakes singular value decomposition of an arbitrary $m \times n$ matrix. Suppose that this matrix is named **Z**. Then singular value decomposition of **Z** leads to computation of the matrices **U**, **S** and **V** where

$$\mathbf{Z} = \mathbf{U}\mathbf{S}\mathbf{V}^t \qquad (15.19.1)$$

In the above equation **U** is an $m \times m$ orthonormal matrix, **V** is an $n \times n$ orthonormal matrix and **S** is a "rectangular diagonal" matrix of dimension $m \times n$ containing the singular values of **Z**. 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 **Z**.

MATSVD is run using the command

```
matsvd matfile umatfile smatfile vtmatfile
```

where

matfile	is a user-supplied matrix file containing an arbitrary rectangular matrix **Z**,
umatfile	contains the SVD-generated **U** matrix,
smatfile	contains a square $j \times j$ **S** (singular value) matrix where j is the smaller of m and n, and
vtmatfile	contains the SVD-generated $n \times n$ $\mathbf{V}^t$ matrix.

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 -1 (i.e. as a list of numbers – see section 2.4 of this manual). This allows the user to much more easily inspect the singular values of **Z** than if they were recorded in a large rectangular matrix of predominantly zero elements.

15.22 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 writes the symmetric matrix, calculated as above.

15.23 MATTRANS

MATTRANS reads a matrix file. It writes another matrix file containing the transpose of the

matrix contained in the first file. It is run using the command

```
mattrans matfile matoutfile
```

where

matfile is the name of a matrix file, and
matoutfile is the name of a new matrix file containing the transpose of the first matrix.

15.24 MATXTXI

MATXTXI calculates $(\mathbf{X}^t\mathbf{X})^{-1}$ where $\mathbf{X}$ 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 matfile matoutfile
```

where *matfile* contains the $\mathbf{X}$ matrix. After completion of MATXTXI execution *matoutfile* contains the matrix $(\mathbf{X}^t\mathbf{X})^{-1}$.

15.25 MATXTXIX

MATXTXIX calculates $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$ where $\mathbf{X}$ 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 matfile matoutfile
```

where *matfile* contains the $\mathbf{X}$ matrix. After completion of MATXTXIX execution *matoutfile* contains the matrix $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$.

15.26 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 this control file; so too is the matrix file itself, using initial parameter values recorded in the PEST control file. A complementary transformation vector file (the second matrix file required by the VECLOG utility) 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 expedites the process of allowing parameter value matrix manipulation to be implemented as part of a model run by PEST during a parameter estimation or predictive analysis process. The model, as run by PEST, will probably need to be upgraded (normally by adding lines to the model batch file) such that one or more of the matrix manipulation utilities documented in the present chapter are run; if any parameters are log-transformed the command to run VECLOG will need to precede all other matrix operations undertaken through this batch 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 the VECLOG utility.

15.27 VEC2PEST

VEC2PEST facilitates the use of matrix manipulation outcomes by PEST. This can be useful where parameters, or parameter projections, must be subjected to constraints imposed by a covariance matrix when maximizing or minimizing a key model prediction as part of a predictive analysis process (see the REGPRED utility documented elsewhere in this manual). 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. a vector) residing in a matrix file. VEC2PEST generates an instruction file with 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 a 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.

The covariance matrix file *covfile* may have been prepared by utility software such as PARAMERR or PREDUNC7. Note that VEC2PEST does not check for the existence of the optionally nominated covariance matrix file. It simply places its name at the appropriate

location within the PEST control file which it generates, opposite the name of the new observation group in the "observation groups" section of this file.

15.28 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 vector 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 writes the new vector to a new matrix file.

VECLOG is run using the command

```
veclog matfile1 matfile2 matoutfile
```

where

matfile1	contains the first vector,
matfile2	contains the second vector, and
matoutfile	is the name of a new matrix file to which the 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.

16. RRF and PAROBS Files

16.1 Introduction

An "RRF file" is a "run results file". This file type is produced by PEST_HP. It contains the parameter values used in a suite of PEST runs, together with model-calculated observation values. Specifications for this file type are provided in section 2.6.

A "PAROBS file" is similar to a parameter value file. However, in addition to the values of parameters, it also contains the values of model outputs calculated using the parameters. In contrast to an RRF file, it contains only one set of parameter values and only one set of model output values. Specifications for a PAROBS fare provided in section 2.7.

16.2 RRFCAT

RRFCAT concatenates two run results files. Before it does so, it checks for compatibility between the files. In particular, it checks that the number of parameters and observations, and the names of the parameters and observations, are the same in each file. RRFCAT can accommodate a different ordering of parameter and observations in the two files; in this case the ordering in the first file is transferred to the final file.

RRFCAT is run using the following command.

```
rrfcat rrfinfile1 rrfinfile2 rrfoutfile
```

where:

rrfinfile1	is an existing run results file,
rrfinfile2	is another existing run results file, and
rrfoutfile	is a new run results file formed by concatenation of the existing run results files.

In concatenating the two files, parameter set indices are renumbered, starting at 1. Also if any model output pertaining to a particular parameter set includes a "null result" (this being marked by any model output value that is less than -1.0e35 for a particular model run), the parameter/model-output set is not transferred to the new run results file.

16.3 RRFCULL

Each parameter and model output set featured in a run results file is characterized by a parameter set index. This integer index is listed directly after the "* parameter set index" header that accompanies each parameter set. Parameter set indices should be sequential and start at 1.

RRFCULL reads a run results file, together with a "culling file". The latter file should contain a list of parameter set indices, one to a line. Optionally, a line in the culling file can commence with a "#" character; the line is then ignored. RRFCULL writes a new run results file. Parameter sets characterized by parameter set indices listed in the culling file are ommitted from the latter file. Parameter sets in this new file are provided with new indices. In accordance with the protocol for a run results file, these indices are sequential, and start at 1.

RRFCULL is run using the following command.

```
rrfcull rrfinfile cullfile rrfoutfile
```

where:

rrfinfile	is an existing run results file,
cullfile	is a culling file, and
rrfoutfile	is the culled run results file.

16.4 RRFCLEAN

RRFCLEAN reads an existing run results file and writes a new one. In writing the new run results file, it performs the following actions on data that is resident in the existing run results file.

- It removes parameter sets for which model output values indicate a failed or abandoned model run. (If any model output value is less than -1.0e35 for a particular parameter set, this is taken as a sign of model run failure or abandonment; this protocol is employed by PEST_HP when it writes a run results file.)
- If data pertaining to the final parameter set is missing from the existing run results file, then no data pertaining to the final parameter set is written to the new run results file.
- If required, parameter set indices are re-numbered, starting at 1 and increasing by 1 for each new parameter set.

RRFCLEAN is run using the command

```
rrfclean rrffile1 rrffile2
```

where:

rrffile1	is the name of an existing run results file, and
rrffile2	is the name of the new run results file written by RRFCLEAN.

16.5 RRF2PAR

RRF2PAR writes a series of parameter value files, based on the contents of a run results file. Parameter value files can be used by programs such as PARREP for conducting a series of model runs, perhaps for the purpose of exploring predictive uncertainty. (If this is the case, the run results file will have been written by a program that has not only calculated sets of parameter values, but has also calculated model output values - probably under calibraton conditions - associated with each of these sets.)

RRF2PAR is run using the command

```
rrf2par rrffile parfilebase index1 index2
```

where:

rrffile	is the name of an existing run results file,
parfilebase	is the filename base of the set of parameter value files which RRF2PAR must write,
index1	is the initial parameter value index, and
index2	is the final parameter value index.

Parameter value indices in a run results file should be arranged in increasing order, starting at 1. Suppose that RRF2PAR is run using the following command.

```
rrf2par file.rrf parfile 10 20
```

Then RRF2PAR reads file *file.rrf* to obtain parameter values. (It ignores model output values). It gathers parameter values associated with parameter set indices 10 to 20 (inclusive), recording these in parameter value files *parfile10.par*, *parfile11,par…parfile20.par*.

16.6 PAROBS2RRF

PAROBS2RRF reads a sequence of PAROBS files. It builds a single run results file in which the contents of all of the PAROBS files are recorded.

PAROBS2RRF is run using the following command.

```
parobs2rrf filebase i1 i2 rrffile
```

where:

filebase	is the filename base of a set of PAROBS files (an extension of "*.parobs*" is assumed),
i1	is the PAROBS file starting index,
i2	is the PAROBS file ending index, and
rrffile	is the run results file which PAROBS2RRF writes.

Suppose, for example, that PAROBS2RRF is run using the command:

```
parobs2rrf case 1 200 outfile.rrf
```

PAROBS2RRF will attempt to read files *case1.parobs*, *case2.parobs…case200.parobs*. It will record parameter values and corresponding model output values that it reads from these files in the run results file *outfile.rrf*. If any of *case*.parobs* files are missing, then PAROBS2RRF will report this to the screen, but will continue its processing of the sequence of PAROBS files. In accordance with the run results file protocol, parameter set indices in the run results file will continue to be sequential, regardless of the missing PAROBS file; hence parameter set indices will not coincide with PAROBS file indices where a PAROBS file is missing. However each set of parameter and model output values can still be linked to a PAROBS file through the "parameter value source" descriptor associated with each parameter set recorded in the run results file.

16.7 RRF2PAROBS

RRF2PAROBS extracts a single set of parameter values and corresponding model output values from a run results file. It writes these to a PAROBS file.

RRF2PAROBS is run using the command

```
rrf2parobs rrffile psetindex parobsfile
```

where:

rrffile	is the name of a run results file,
parsetindex	is a parameter set index cited in this file, and
parobsfile	is the name of the PAROBS file which RRF2PAROBS must write.

16.8 RRFAPPEND

RRFAPPEND appends parameter and model output values read from a PAROBS file to the end of a run results file. However, it only does this if the names and ordering of parameters and model outputs in the PAROBS file are the same as those in the run results file. The parameter set index provided to the new dataset is obtained by incrementing the last parameter set index found in the run results file.

RRFAPPEND is run using the command

```
rrfappend parrobsfile rrffile
```

where:

parrobsfile is the name of an existing PAROBS file, and

rrffile is the name of an existing run results file.

If RRFAPPEND detects any inconsistencies between parameter names and ordering in the PAROBS and run results files, it ceases execution with an error message. It does the same if it encounters an error in either file. If it encounters an unexpected end to the run results file, it will still append the contents of the PAROBS file to the end of this file. However it will leave two empty lines between the current end of the run results file and the beginning of the appended data.

If the run results file to which RRFAPPEND must append parameter and model output data does not exist, then RRFAPPEND creates this file and writes a header to it.

16.9 MULPAROBSTAB

MULPAROBSTAB performs a very similar role to that of the MULPARTAB utility. However it operates on a suite of PAROBS files rather than on a suite of parameter value files. Because of this, it writes two output files, one in which the values of parameters are tabulated and one in which the values of model outputs are tabulated.

MULPAROBSTAB is run using the command

```
mulparobstab parobsfile listfile outfile1 outfile2
```

where

parobsfile is a generic PAROBS filename,
listfile contains a list of integer indices,
outfile1 is the name of a tabular output file in which parameter values are recorded, and
outfile2 is the name of a tabular output file in which model output values are recorded.

16.10 RRF2TAB

RRF2TAB performs a simular role to MULPAROBSTAB, in that it produces two tabular data files. However it reads parameter and model output values from a run results file rather than from a series of PAROBS files. It is run using the command

```
rrf2tab rrffile outfile1 outfile2
```

where

rrffile is a run results file,

outfile1 is the name of a tabular output file in which parameter values are recorded, and

outfile2 is the name of a tabular output file in which model output values are recorded.

16.11 RRF2CSV

RRF2CSV performs a similar role to that of RRF2TAB. It reads a run results file. However parameter and model output values listed in that file are recorded in CSV files rather than in tabular files. It is run using the command

```
rrf2tab rrffile outfile1 outfile2
```

where

rrffile is a run results file,
outfile1 is the name of a CSV file in which parameter values are recorded, and
outfile2 is the name of a tabular output file in which model output values are recorded.

It is suggested that both of files written by RRF2CSV be provided with an extension of "*.csv*".

16.12 PARREP_RRF

PARREP_RRF performs a similar function to PARREP in that it writes a new PEST control file which is identical to an existing PEST control file, except for the fact that parameters in the new PEST control file have new values. For PARREP, those values are read from a parameter value file. For PARREP_RRF, new parameter values are read from a run results file.

PARREP_RRF is run using the command

```
parrep_rrf rrffile parsetindex pestfile1 pestfile2
```

where

rrffile is the name of a run results file,

parsetindex (an integer) is a parameter set index cited in this file,

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

pestfile2 is the name for the new PEST control file.

16.13 RRFCALCPSI

16.13.1 General

RRFCALCPSI reads a run results file. As well as reading a run results file, RRFCALCPSI reads a PEST control file. This file may be the same as that which PEST_HP used for the run on which it actually wrote the RRF file. Alternatively, it may be different. If it is different, it must employ the same parameters and the same observations as those featured in the run results file. However any or all of the following can be different between the PEST control file whose name is supplied to RRFCALCPSI and that on which production of the run results file was based:

- the ordering of parameters and observations;
- the weights and/or covariance matrices associated with observations;
- the measured values assigned to observations;
- the groups to which observations are assigned;
- whether or not prior information is employed, and the nature of that prior information;
- the mode in which PEST_HP was run (for example "estimation" or "regularisation").

For each parameter and model output set that it finds in the run results file, RRFCALCPSI calculates the objective function pertaining to each observation group, as well as the total objective function. Included in these objective functions are contributions made by prior information. It is important to note, however, that in calculating these objective function components, RRFCALCPSI employs weights, covariance matrices, and prior information equations featured in the PEST control file whose name is provided to it, and not those in the original PEST control file used by PEST_HP (or any other software) to write the run results file. Objective functions calculated by RRFCALCPSI are tabulated in a columnar data file that is easily imported into a spreadsheet package such as EXCEL. Also recorded is the parameter set index from the run results file, as well as the reason for each model run; the latter is written in the "parameter values source" component of each run record in the run results file.

16.13.2 Using RRFCALCPSI

RRFCALCPSI is run using the following command. (Type its name at the command-line prompt if you forget, and RRFCALCPSI will remind you.)

```
rrfcalcpsi rrffile pestfile psifile
```

where

rrffile	is the name of a run results file,
pestfile	is the name of a PEST control file, and
psifile	is a file in which objective functions will be recorded.

Note the following:

- The run results file is assumed to have an extension of "*.rrf*". If this is omitted, RRFCALCPSI will add this extension to the name which you provide.
- The PEST control file must have an extension of "*.pst*". If this is omitted, RRFCALCPSI will add this extension to the name which you provide.
- The file written by RRFCALCPSI can have any extension. Its extension should be included with its name when issuing the above command.

As is described in specifications for a run results file (see section 2.6 of this document), model output values of -1.11E35 in the run results file indicate a failed model run, while model output values of -1.22E35 indicate an abandoned model run. RRFCALCPSI calculates objective function values of -1.11E35 and -1.22E35 for these occasions.

If any prior information is present in the PEST control file whose name is supplied to RRFCALCPSI, objective functions corresponding to this prior information are calculated, together with those pertaining to model outputs. Prior information outcomes do not depend on model outputs. They only depend on parameter values; these are read from the run results file.

If the PEST control file whose name is supplied to RRFCALCPSI instructs PEST to run in

"regularisation" mode, then the regularisation weight factor used in calculation of the objective functions associated with regularisation groups is equal to the initial weight factor (i.e. WFINIT) supplied in the "regularisation" section of the PEST control file.

If the PEST control file whose name is supplied to RRFCALCPSI instructs PEST to run in "pareto" mode, then no weight factors are employed. Weights and covariances matrices recorded in the PEST control file are used directly.

16.14 RRF2JCO

16.14.1 General

RRF2JCO reads a run results file. It calculates a binary Jacobian matrix file (i.e. a JCO file) based on parameters and model outputs that are recorded in this file.

16.14.2 Theory and Concepts

Let the vector **h** represent model outputs under calibration conditions. Thus, for a linearized model:

$$\mathbf{h} = \mathbf{Z}\mathbf{k} \tag{16.14.1}$$

The previous equation can be expanded as follows.

$$\begin{bmatrix}\mathbf{h}\\ \mathbf{k}\end{bmatrix} = \begin{bmatrix}\mathbf{Z}\\ \mathbf{I}\end{bmatrix}\mathbf{k} \tag{16.14.2}$$

Suppose now that random realizations of parameters **k** are generated using the variance-covariance matrix C(**k**). The covariance matrix C_{hk} relating model outputs to random parameters can be calculated as follows.

$$C\left(\begin{bmatrix}\mathbf{h}\\ \mathbf{k}\end{bmatrix}\right) = \begin{bmatrix}C_{hh} & C_{hk}\\ C_{kh} & C_{kk}\end{bmatrix} = \begin{bmatrix}\mathbf{Z}\\ \mathbf{I}\end{bmatrix} C(\mathbf{k}) \begin{bmatrix}\mathbf{Z}^t & \mathbf{I}\end{bmatrix} = \begin{bmatrix}\mathbf{Z}C(\mathbf{k})\mathbf{Z}^t & \mathbf{Z}C(\mathbf{k})\\ C(\mathbf{k})\mathbf{Z}^t & C(\mathbf{k})\end{bmatrix} \tag{16.14.3}$$

so that

$$C_{hk} = \mathbf{Z}C(\mathbf{k}) \tag{16.14.4}$$

C_{hk} can be calculated empirically based on the outputs of many model runs that employ samples of C(**k**). Based on these model outputs, an approximation to the sensitivity matrix (i.e. the Jacobian matrix) **Z** can be calculated from equation 16.14.4 as

$$\mathbf{Z} = C_{hk}C^{-1}(\mathbf{k}) \tag{16.14.5}$$

RRF2JCO calculates $C^{-1}(\mathbf{k})$ using the equation

$$C^{-1}(\mathbf{k}) = \mathbf{U}\mathbf{S}^{-1}\mathbf{U}^t \tag{16.14.6}$$

where **U** and **S** are obtained through singular value decomposition of C(**k**). Prior to its inversion, C(**k**) can optionally be calculated empirically from the values of parameters featured in the run results file. If it is singular, this does not affect its inversion through 16.14.6; zero-valued singular values are simply ignored.

RRF2JCO calculates the value of element (*i*,*j*) of C_{hk} from the outcomes of *n* model runs based on *n* random parameter sets using the equation:

$$C_{hk}(i,j)=\frac{\sum_{m=1}^{n}(h_{im}-\underline{h}_i)(k_{jm}-\underline{k}_j)}{n-1} \qquad (16.14.7)$$

where $\underline{h}_i$ is calculated as:

$$\underline{h}_i=\frac{\sum_{m=1}^{n}h_{im}}{n} \qquad (16.14.8)$$

and $\underline{k}_j$ is calculated as:

$$\underline{k}_j=\frac{\sum_{m=1}^{n}k_{jm}}{n} \qquad (16.14.9)$$

If C_{kk} is calculated empirically, a similar equation is employed. Optionally the user can supply values for $\underline{h}_i$ and $\underline{k}_j$ for use in equation 16.14.7 (and in a similar equation pertaining to C_{kk}) him/herself.

16.14.3 Using RRF2JCO

RRF2JCO is run using the following command. (Type its name at the command-line prompt if you forget; and RRF2JCO will remind you.)

```
rrf2jco pestfile mfile/nul/pst ufile/nul rrffile jcofile [sthresh]
```

where

pestfile	is the name of a PEST conrol file,
mfile/nul/pst	is the name of an optional mean value file (in PAROBS format),
ufile/nul	is the name of an optional parameter uncertainty file,
rrffile	is the name of a run results file,
jcofile	is the name of the Jacobian matrix file which RRF2JCO writes, and
sthresh	is an optional singular value truncation ratio.

Immediately on commencement of execution, RRF2JCO reads a PEST control file. While RRF2JCO does not use parameter or observation values from this file, it must nevertheless read a PEST control file in order to obtain the transformation status of each parameter. As is described elsewhere in PEST documentation, if a parameter is log-transformed then sensitivities recorded in the JCO file must pertain to the log (to base 10) of the parameter.

If the PEST conrol file which RRF2JCO reads contains prior information, this is represented in the JCO file which RRF2JCO writes.

RRF2JCO obtains the variance-covariance matrix C(**k**) in either of two ways. If random parameter values on which model runs were based were generated using parameter uncertainties specified in a parameter uncertainty file (see section 2.5 of this manual), C(**k**) can be calculated from the contents of this uncertainty file. Alternatively, RRF2JCO calculates an empirical C(**k**) matrix from the parameter values which appear in the run results file. Specifiy the name of a parameter uncertainty file as RRF2JCO's third command line argument if you wish that C(**k**) be obtained in the former way; specify "nul" for the name of

the parameter uncertainty file to choose the latter option.

Where a covariance matrix is calculated empirically, it may not be invertible. In fact it certainly will not be invertible if the number of random parameters sets on which calculation of an empirical C(**k**) is based is less than the number of elements of **k**. In this case singular values featured in the **S** matrix of equation 16.14.6 must be truncated before being inverted. RRF2JCO will automatically truncate at (*n*-1) singular values, where *n* is the number of random parameter sets featured in the run results file. However, through appropriate choice of the *sthresh* command-line parameter, RRF2JCO can be asked to truncate at that point where the ratio of a particular singular value to the first singular value is equal *sthresh*. RRF2JCO employs a default *sthresh* value of 5.0e-7.

Optionally, RRR2JCO can read mean parameter and model output values used in calculation of the C_{hk} empirical covariance matrix (see equation 16.14.7), and the optional empirical C_{kk} covariance matrix, from a PAROBS file. Provide the name of the PAROBS file as the second RRF2JCO command line argument; alternatively supply a name of "nul" for the PAROBS file for mean model output and parameter values to be calculated using equations 16.14.8 and 16.14.9.

RRF2JCO provides one further alternative for calculation of mean values. This is activated by providing the string "pst" instead of the name of a PAROBS file or "nul" for the second RRF2JCO command line argument. With this argument, RRF2JCO uses parameter values listed in the PEST control file cited as RRF2JCO's first command line argument as mean parameter values. It searches for corresponding model output values in the run results file which is cited as its fourth command line argument. If the run results file does not include the parameter set cited in the PEST conrol file, RRF2JCO automatically sets the second command line argument to "nul"; hence parameter and observation means are computing by averaging values in the run results file.

17. Support for the PEST++ Suite

17.1 Enhanced Jacobian Matrix Files

An "enhanced Jacobian matrix file" does not necessarily hold a Jacobian matrix. Rather, it holds an arbitrary two-dimensional array of double precision numbers. This array is stored in binary format, with zero-valued elements omitted. This constitutes an efficient form of storage for matrices which are sparse. The name "enhanced Jacobian matrix file" arises from the fact that its compressed storage protocol is not unlike that used by PEST for storing a Jacobian matrix. An important point of similarity with the PEST Jacobian matrix protocol is that each row and column of the matrix has a name. These names are stored in the file following the matrix. In contrast to names stored in PEST matrix files however, the names of rows and columns associated with matrices stored in an enhanced Jacobian matrix files can be up to 200 characters in length.

17.2 EJCOL2VEC

EJCOL2VEC performs the same role for an enhanced Jacobian matrix file as that which JCOL2VEC performs for a Jacobian matrix file. That is, it extracts a column of the matrix stored in the file and records that column as a one-dimensional column matrix in PEST matrix file format. EJCOL2VEC records the full names of matrix rows and columns in its output file, even if they are over 20 characters in length, despite the fact that this violates PEST matrix file protocol; under these circumstances the matrix file produced by EJCOL2VEC will not be readable by other PEST matrix file utility programs.

EJCOL2VEC is run using the command:

```
ejcol2vec jcbfile colname matfile
```

where

jcbfile	is the name of an enhanced Jacobian matrix file (with extension "*.jcb*)",
colname	is the name of a column featured in that file, and
matfile	is the name of the matrix file to which the extracted column of the matrix is to be written as a vector.

17.3 EJROW2VEC

EJROW2VEC performs the same role for an enhanced Jacobian matrix file as that which JROW2VEC does for a Jacobian matrix file. That is, it extracts a row of the matrix and stores that row as a one-dimensional column matrix in PEST matrix file format; in doing so, the matrix row becomes a column vector, so that it is effectively transposed by this operation. EJROW2VEC records the full names of matrix rows and columns in its output file, even if they are over 20 characters in length, despite the fact that this violates PEST matrix file protocol; under these circumstances the matrix file produced by EJROW2VEC will not be readable by other PEST matrix file programs.

EJROW2VEC is run using the command:

```
ejrow2vec jcbfile rowname matfile
```

where

jcbfile	is the name of an enhanced Jacobian matrix file (with extension "*.jcb*)",

rowname is the name of a row featured in that file, and
matfile is the name of the matrix file to which the extracted row of the matrix is to be written as a vector.

17.4 EJROWCOL

EJROWCOL reads an enhanced Jacobian matrix file. It lists, to a user-supplied file, the names of the rows and columns of the matrix that is stored in this file.

EJROWCOL is run using the command:

```
ejrow2vec jcbfile outfile
```

where

jcbfile is the name of an enhanced Jacobian matrix file (with extension "*.jcb*)", and
matfile is the name of the file to which matrix row and column names are listed.

17.5 GETCSVCOL

Program GETCSVCOL extracts a column of numbers from a CSV file. It lists the numbers that it extracts in a user-nominated text file. Depending on directives that are supplied on its command line, this list could comprise the second column of a PEST parameter value file; alternatively, it could comprise the first and only column of a file that contains only a list of numbers.

GETCSVCOL receives instructions through its command line. It is run as follows:

```
getcsvcol csvfile ncol outfile [/h] [/p]
```

where:

csvfile is the name of a CSV file;

ncol+1 identifies the column number whose contents GETCSVCOL must read; and

outfile is the file in which it must record these contents.

If the "/h" switch is supplied on its command line, then the first row of the CSV file that GETCSVFILE reads is ignored, for this row is presumed to contain column headers. The header to the user-nominated column of interest is not transferred to the data file that GETCSVCOL writes.

If the "/p" switch is supplied on its command line, then GETCSVCOL records the contents of the user-specified CSV file column in a PEST parameter value file. Precision and decimal point protocols for this file are "single point", while all parameter scales and offsets are presumed to be 1.0 and 0.0 respectively. The names of parameters that feature in the first column of the parameter value file are read from the first column of the CSV file (which is thereby assumed to be comprised of row names).

Note the way in which the second command-line argument of GETCSVCOL is used. Suppose that *ncol* is supplied as 1. Then GETCSVCOL extracts numbers from the 2nd column of the file (presumed to be the first data column of the file, as the first column is presumed to contain row names).

Note also, that GETCSVCOL will cease execution with an error message unless all of the elements in the column of the CSV file that it reads can be interpreted as numbers.

17.6 ADDCSVCOL

As the name implies, ADDCSVCOL performs the opposite role to GETCSVCOL. It adds a column of data to an existing CSV file provided that certain conditions are met. Alternatively, it creates a new CSV file.

ADDCSVCOL is run using the following command:

```
addcsvcol obsfile csvfile headertext
```

where:

obsfile is an "observation value file" written by INSCHEK or some other program;

csvfile is the name of a new or existing CSV file; and

headertext is the text that will head the new CSV file column.

An observation value file has two columns of data with no header. The first column contains observation names while the second column contains observation values. INSCHEK writes such a file if asked to read a model output file.

If the CSV file that is cited on the ADDCSVCOL command line does not exist, then ADDCSVCOL writes a two-column CSV file based on the contents of the observation value file. In contrast to an observation value file, this new CSV file has column headers. The header of the first column is "obsname". The header of the second column is the third entry on the ADDCSVCOL command line.

If the CSV file that is cited on the ADDCSVCOL command line does exist, then ADDCSVCOL checks that the names in the first column of this file are the same as those in the first column of the observation value file that it reads. If this is not the case, it ceases execution with an appropriate error message. Otherwise it adds a new column to the CSV file by transferring the contents of the second column of the observation data file to this new column. This data column is headed by the *headertext* character string supplied as the third item on the ADDCSVCOL command line.

17.7 JCB2RRF

17.7.1 General

PESTPP-IES provides the option to store realizations of parameter and corresponding model output values in enhanced Jacobian file format. Suppose that the name of the PEST control file on which PESTPP-IES's operations are based is named *case.pst*. Then, if this option is selected, PESTPP-IES records files named *case.N.par.jcb* and *case.N.obs.jcb* at the end of iteration *N*. These files contain (adjusted) parameter realizations and corresponding model outputs respectively; values of all parameters and model outputs for all realizations are stored along with the realization index in these files. The realization index is either "base" or an integer; realization indices other than "base" start at 0. Note however that they may not be stored in sequential order in JCB files; however the order in which they are stored is the same in paired parameter and observation JCB files which pertain to the same iteration number.

PESTPP-IES stores parameter realizations and corresponding model outputs in JCB format if the string:

```
++ ies_save_binary(true)
```

appears in the problem-defining PEST control file. If this is the case, then PESTPP-IES does

not record this same data in iteration-specific CSV files.

While storage of PESTPP-IES outcomes in CSV files is useful in some ways, it can also be problematic. If there are many parameters and observations, these files can be very wide, and hence difficult to read and process. While transposition of rows and columns in these files can be achieved by using the

```
++ ies_csv_by_reals(false)
```

specifier in the problem-defining PEST control file, this too has its problems. It requires that the whole contents of each CSV file be held in storage if all paired parameter/observation realizations are to be extracted from these files and written in another format (as is the task of JCB2RRF).

Reading parameter and observation realizations from paired JCB files overcomes all of these problems.

As is documented in section 2.6 of this manual, a "run results" file provides another mechanism for storage of multiple parameter sets and corresponding model outputs. Use of this file type also overcomes the problems associated with CSV file storage which were outlined above. (Obviously, however, parameter and corresponding model output ensembles are not as easily viewed in a spreadsheet when stored in a run results file.)

A number of utilities that are documented in the present manual can extract and process model parameters and outcomes stored in a run results file. These include RRFCALCPSI (which can associate objective functions with model outcomes) and RRF2JCO (which can calculate an approximate Jacobian matrix from parameter and corresponding model output realizations stored in a run results file.)

17.7.2 Using JCB2RRF

JCB2RRF is run using the command

```
jcb2rrf jcbfilebase rrffile
```

where

jcbfilebase	is the filename base of two JCB files, one of which contains parameter realizations and the other of which contains corresponding realizations of model outputs, and
rrffile	is the name of a run results file in which data from the two JCB files is recorded.

Caution must be exercised when supplying *jcbfilebase*. As was stated above, suppose that this is supplied as "*case.8*". JCB2RRF then tries to open two JCB files, these being *case.8.par.jcb* and *case.8.obs.jcb*. If it fails to find or open one of these files, it ceases execution with an appropriate error message.

The user-supplied *rrffile* filename should have an extension of "*.rrf*". If it does not, then JCB2RRF adds it automatically.

17.8 JCB2RRF1

JCB2RRF1 is identical to JCB2RRF except for the fact that it performs a more limited set of checks on the JCB files that it reads than does JCB2RRF. In particular, it does not require realization name compatibility between the two JCB files. Nor does it require that realizations be named "base" or an integer value. However, it does require consistency of the number of

realizations that are stored in the two JCB files.

JCB2RRF1 pairs parameter and observation realization names in order of their appearance in their respective files.

17.9 JCB2CSV

JCB2CSV writes a matrix contained in a JCB file as a comma-delimited file. As such, the file can be exported into EXCEL for viewing. It is run using the command:

```
jcb2csv jcbfile csvfile transpose
```

where

jcbfile	is the name of a JCB file (a filename extension of "*.jcb*" is required),
csvfile	is the name of a CSV file (a filename extension of "*.csv*" is required), and
transpose	must be either "t" or "nt".

If *transpose* is supplied as "t" then the matrix suppled in the JCB file is transposed before it is recorded in the CSV file. If it is supplied as "nt", it is not transposed. Where a JCB file contains parameter or model output realizations recorded by PESTPP-IES, "t" is usually the better option as the number of parameters or model outputs normally far exceeds the number of realizations.

The first row of the CSV file written by JCB2CSV contains column or row names (depending on whether *transpose* is set to "nt" or "t" respectively). The first column of the CSV file written by JCB2CSV contains row or column names (depending on whether *transpose* is set to "nt" or "t" respectively).

17.10 CSV2JCB

CSV2JCB performs the opposite task to that performed by JCB2CSV. It reads a CSV file produced by PESTPP-IES and re-writes it as a JCB file. With PESTPP-IES outputs written in this format, they can be subjected to processing provided by some of the utility programs documented herein which read JCB files, but not CSV files.

CSV2JCB is run using the command:

```
csv2jcb csvfile jcbfile
```

where

csvfile	is the name of a CSV file written by PESTPP-IES (a filename extension of "*.csv*" is required), and
jcbfile	is the name for a JCB file (a filename extension of "*.jcb*" is required).

17.11 JCB2PAR

JCB2PAR reads a JCB file written by PESTPP-IES or RSI_HP containing parameter values comprising parameter ensembles. Normally the name of this file is *case.N.par.jcb* where the history-matching process is based on file *case.pst*, and *N* is the iteration number.

JCB2PAR operates in two ways, depending on the way that it is run. In the first of these ways it extracts a single row of the JCB file (this pertaining to a realization) and records parameter values comprising the nominated realization in a single parameter value file named according to the user's choice. Alternatively, it can extract data comprising all rows of this file, and then

write a series of parameter value files, one for each realization, thus collectively listing the contents of all parameter ensembles.

To extract a single ensemble from a JCB file, run JCB2PAR as follows.

```
jcb2par jcbfile realname parfile
```

where

jcbfile	is the name of a JCB file (for which a filename extension of "*.jcb*" is required),
realname	is the name of a realization, and
parfile	is the name of a parameter value file (for which an extension of "*.par*" is required).

Note that realization names are integers, starting at zero, or "base".

Alternatively, JCB2PAR can be run using the following command.

```
jcb2par jcbfile all parfilebase
```

where

jcbfile	is the name of a JCB file (for which a filename extension of "*.jcb*" is required),
all	must be written as "all", and
parfilebase	is a filename base.

In this latter case JCB2PAR writes a series of PAR files, one for each realization recorded in the JCB file. These parameter value files are named *parfilebase_realname.par* where *parfilebase* is the user-supplied filename base of the parameter value files, and *realname* is a realization name which, as stated above, is either an integer or "base".

As is described elsewhere in this manual, parameter values recorded in a parameter value file can be inserted into a PEST control file using the PARREP utility. If NOPTMAX is set to zero in this PEST control file, the model can be run using these parameter values.

17.12 JCB2AVEPAR

JCB2AVEPAR reads a JCB file written by PESTPP-IES or RSI_HP containing parameter values comprising parameter ensembles. Normally the name of this file is *case.N.par.jcb* where the history-matching process is based on the PEST control file *case.pst*, and *N* is the iteration number.

JCB2AVEPAR is somewhat similar to JCB2PAR in that it writes a parameter value file (i.e. PAR file) based on the contents of the JCB file. However the PAR file which it writes contains parameter values which, for each parameter, is the average of all values recorded for that parameter in the JCB file. Averaging is done over native parameter values or the logs of parameter values, this depending on the PARTRANS value for that parameter in the PEST control file on which the JCB file is based. If PARTRANS is set to "log" then averaging is done over the logs of parameters (and then converted back to its native value before being recorded in the PAR file). If PARTRANS for a parameter is set to "none", then averaging is done over the native values of that parameter as recorded in the JCB file. If the parameter is fixed in the PEST control file, then its initial value as recorded in the PEST control file is transferred to the PAR file which JCB2AVEPAR writes. If the parameter is tied in the PEST control file, then the value recorded for that parameter in the PAR file maintains the same ratio with the parent parameter as that provided in the PEST control file.

JCB2AVEPAR is run as follows:

```
jcb2par jcbfile pestfile parfile
```

where

jcbfile	is the name of a JCB file (for which a filename extension of "*.jcb*" is required),
pestfile	is the name of a PEST control file (for which a filename extension of "*.pst*" is required), and
parfile	is the name of a parameter value file (for which an extension of "*.par*" is required).

17.13 JCBCOMB

JCBCOMB combines the contents of two JCB files. It extracts user-specified rows from a first JCB file, and user-specified rows from a second JCB file. It writes a new JCB file which features the rows from the first of these files followed by the rows from the second of these files.

Typical prompts and responses are as follows:

```
Enter name of first JCB file: random1.jcb
  Enter lowest row number to transfer [1 - 800]: 1
  Enter highest row number to transfer [1 - 800]: 50
  Enter prefix for new row names (5 chars or less): a

Enter name of second JCB file: random2.jcb
  Enter lowest row number to transfer [1 - 800]: 1
  Enter highest row number to transfer [1 - 800]: 50
  Enter prefix for new row names (5 chars or less): b

Enter name for new JCB file: random3.jcb

- binary matrix file random1.jcb read ok.
- binary matrix file random2.jcb read ok.
- binary random3.jcb written ok.
```

The two JCB files whose contents are combined must have the same number of columns. If they do not, JCBCOMB will cease execution with an error message. Ideally the names of these columns should be the same; they are often the names of parameters. If this is not the case, JCBCOMB will issue a warning; column names from the first JCB file will be transferred to the file that JCBCOMB actually writes.

In responding to JCBCOMB's prompts, rows are denoted by number. Numbers start at 1. This is to be distinguished from numbers which may actually comprise the names of rows in JCB files. Rows are renamed as they are transferred to the new JCB file. A user-suppled prefix is affixed to each set of names. This prefix does not need to be a number.

17.14 CSV2PAR

CSV2PAR reads a CSV file written by PESTPP-IES that lists parameter values corresponding to all realizations that comprise a parameter ensemble. Generally this file is named *case.N.par.csv* where *case* is the filename base of the PEST control file and *N* is the iteration number. PESTPP-IES adopts one of two protocols when writing this file. In one of these protocols (the default) parameters are arranged in rows, while in the other protocol

parameters are arranged in columns. In the first case the CSV file has *NPAR*+1 columns and *NREAL*+1 rows, while in the second case the CSV file possesses *NREAL*+1 columns and *NPAR*+1 rows, where NPAR is the number of parameters and NREAL is the number of realizations featured in the PESTPP-IES history-matching process. The extra row/column contains parameter and realization names.

CSV2PAR is designed to read very large CSV files. These files can contain more rows and columns than can be handled by Microsoft EXCEL.

Unlike many other PEST utility programs, CSV2PAR optionally (see below) receives information through user-supplied responses to its prompts. As there are five such prompts, this is a little more convenient than supplying this information through the command line. CSV2PAR's first prompt is:

```
Enter name of CSV file:
```

Provide the name of a CSV file written by PESTPP-IES. CSV2PAR reads this file, holding its entire contents in its memory. It next asks:

```
Extract param vals from a row or column of this file? [r/c]:
```

Respond with "r" or "c" as appropriate. If the response to this prompt is "r", CSV2PAR then asks:

```
Enter name of row:
```

Alternatively, it asks:

```
Enter name of column:
```

The name supplied in response to this prompt must be the name of a PESTPP-IES realization. This is generally a number or "base". CSV2PAR's next prompt is:

```
Enter name for output parameter value file:
```

Finally:

```
Enter name for data file (<Enter> if none):
```

The parameter value file written by CSV2PAR can be used by many other programs of the PEST suite. For example, program PARREP can populate a PEST control file with parameter values that it reads from this file.

The data file that is written by CSV2PAR contains all of the numbers in the requested row or column of the CSV file listed one number to a line. This can be read by a program from the MODFLOW family using its OPEN/CLOSE functionality.

Alternatively, CSV2PAR can receive information from the command line. In order to do this, run it using the following command.

```
csv2par csvfile "r" or "c" realname parfile [datfile]
```

where

csvfile	is the name of the CSV file that it must read,
"r" or "c"	determines whether realizations are stored in rows or columns,
realname	is the name of the realization that it must read,
parfile	is the name of the parameter value file that it must write, and
datfile	is the name of the optional data file that it must write.

17.15 RANDOBS

17.15.1 General

RANDOBS generates random realizations of model outputs for the use of the PESTPP-IES ensemble smoother. These realizations can be saved in a comma-delimited file (i.e. a CSV file) or in an enhanced Jacobian matrix file (i.e. a JCB file). In the former case observation realizations can be recorded one to a row, or one to a column.

RANDOBS reads an "observation weights file". This can be any text file that has an "observation data" section. This can include a PEST control file. The observation data section must start with the string "* observation data". Any line beginning with a "*" character terminates this section; alternatively, the end of the file terminates the section.

The "observation data" section must have at least three columns of data. The first column is comprised of observation names, the second column is comprised of observation values, while the third column is comprised of observation weights. RANDOBS assumes that weights are equal to the inverse of the standard deviation of measurement noise. It uses this standard deviation to generate a random value for each observation centred on the value of that observation recorded in the observation weights file; a normal distribution is assumed. Note however that if the weight is zero for a particular observation, then the standard deviation for that observation is assumed to be zero, so that its value in all realizations is the same, i.e. the value of the observation itself. Optionally, all observation weights can be multiplied by a factor before being used in the above manner for random realization generation.

17.15.2 Running RANDOBS

RANDOBS commences execution with the prompt:

```
Enter name of observation weights file:
```

The specifications for this file are outlined above; optionally, this can be a PEST control file. Next RANDOBS asks:

```
Enter factor to apply to all weights:
```

Enter a value that is greater than zero. All weights found in the observation weights file are multiplied by this value.

RANDOBS's next prompt is:

```
How many realizations to generate:
```

to which the response should be a number greater than zero. Next it asks:

```
Include initial observations as base realization? [y/n]:
```

If the answer to this question is "y", then the first realization will be comprised of observation values read from the observation weights file.

Next RANDOBS prompts for the name of the file that it must write:

```
Enter name for output file:
```

The extension of this file must be "*.csv*" or "*.jcb*". In the former case RANDOBS records realizations in a CSV file. In the latter case it records realizations in a JCB file, with one realization per row of this file. For a CSV file, however, the assignment of rows and columns is determined by the user. RANDOBS asks:

```
Realizations are rows or columns in csv file? [r/c]:
```

After prompting for the value of a random number seed (an integer greater than zero),

```
Enter integer random number seed (<Enter> if default):
```

RANDOBS calculates realizations and records these in its output file.

18 Data Space Inversion

18.1 DSI1

18.1.1 Theory

Note: See documentation for DSI2 before using DSI1. In general, DSI2 is the best option for data space inversion. However some of the theory behind data space inversion is explained in documentation for DSI1.

18.1.1.1 General

"DSI" stands for "data space inversion". The purpose of DSI1 is to undertake "history matching" without actually estimating parameters. It is able to condition (and thereby reduce the uncertainty of) a user-specified prediction from its uncertainty calculated on the basis of the prior parameter probability distribution. It accomplishes this using the outcomes of a suite of model runs which span both the calibration and prediction periods. When undertaking these runs, the model employs parameters which are sampled from the prior parameter probability distribution.

18.1.1.2 Conditioning

The theory behind data space inversion, as undertaken by DSI1, is now briefly explained. Part of the following explanation is extracted from Doherty (2015). For an alternative methodology and explanation of the theory, see Sun and Durlofsky (2017).

The DSI1 program implements a quick and approximate version of the Sun and Durlofsky methodology. More comprehensive DSI-type analysis is available from DSI2.

Let $\mathbf{x}$ be a random vector. Suppose that we partition $\mathbf{x}$ into the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ as follows:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \tag{18.1.1}$$

Let the mean of $\mathbf{x}$ be denoted as:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \tag{18.1.2}$$

The prior covariance matrix $C(\mathbf{x})$ of $\mathbf{x}$ can be partitioned according to the partitioning of $\mathbf{x}$ into $\mathbf{x}_1$ and $\mathbf{x}_2$ as:

$$C(\mathbf{x}) = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{18.1.3}$$

If variability of $\mathbf{x}$ is described by a multinormal distribution, then the marginal distribution of $\mathbf{x}_1$ is completely characterized by $\boldsymbol{\mu}_1$ and $\mathbf{C}_{11}$. Suppose now that we acquire perfect knowledge of values taken by elements of the $\mathbf{x}_2$ vector. This modifies the probability distribution of $\mathbf{x}_1$. Let $\boldsymbol{\mu}_1'$ and $\mathbf{C}'_{11}$ specify the mean and covariance matrix of this modified distribution (i.e. the posterior mean and covariance matrix of $\mathbf{x}_1$). It can be shown (see, for example, Koch, 1999) that:

$$\boldsymbol{\mu}_1' = \boldsymbol{\mu}_1 + \mathbf{C}_{12}\mathbf{C}^{-1}_{22}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \quad (18.1.4)$$

and that:

$$\mathbf{C}'_{11} = \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{C}^{-1}_{22}\mathbf{C}_{21} \quad (18.1.5)$$

For the DSI1 program described herein, the $\mathbf{x}_1$ vector has just one component, namely a prediction of interest. Meanwhile $\mathbf{x}_2$ is comprised of measurements which collectively constitute a calibration dataset. That is, it is comprised of calibration-equivalent model outputs to which measurement noise has been added. The covariance matrix $\mathbf{C}_{22}$ which describes the prior variability of these outputs is obtained empirically by running the model a user-specified number of times using parameter sets which are sampled from the prior parameter probability distribution; a random realization of measurement noise is added to each such model output. The prior mean model output vector $\boldsymbol{\mu}_2$ is calculated from these model outputs, as are the mean $\boldsymbol{\mu}_1$ and prior uncertainty variance $\mathbf{C}_{11}$ of the model prediction of interest. The values of measurements comprising the calibration dataset (i.e. the elements of $\mathbf{x}_2$) are supplied by the user. DSI1 uses equation 18.1.4 to calculate the posterior expected value $\boldsymbol{\mu}_1'$ of the prediction which comprises the single element of $\mathbf{x}_1$. It uses equation 18.1.5 to calculate the posterior variance of this prediction, this comprising the single element of the matrix $\mathbf{C}'_{11}$.

As mentioned above, DSI1 fills the $\mathbf{C}_{22}$ and $\mathbf{C}_{12}$ matrices which appear in the above equations by undertaking model runs based on an ensemble of random realizations of parameters sampled from the prior parameter probability distribution. There is no limit to the number of parameters which the model can employ, for parameters are not featured in the above equations. Model output variances and covariances are calculated from the ensemble of model outputs using standard formulae for calculation of variance and covariance (possibly after transformation to enhance multinormality of these outputs).

18.1.1.3 Obtaining the inverse of C_{22}

Where observations comprising a calibration dataset outnumber realizations used for covariance calculation (as is usually the case), the $\mathbf{C}_{22}$ matrix is not of full rank. Hence its inverse $\mathbf{C}^{-1}_{22}$ does not exist. An approximation to this inverse can be calculated using truncated singular value decomposition. However where the number of observations is large, the $\mathbf{C}_{22}$ matrix is very large indeed, for its dimensions are *NOBS*×*NOBS* where *NOBS* is the number of non-zero-weighted observations featured in a calibration dataset. The computational effort required to calculate an approximation to $\mathbf{C}^{-1}_{22}$ may therefore become prohibitive. DSI1 overcomes this problem by undertaking singular value decomposition on a matrix whose dimensions are NREAL×NREAL rather than NOBS×NOBS, where NREAL is the number of realizations of parameters (i.e. the number of model runs) used to compute $\mathbf{C}_{22}$; as already stated, this is normally far less than the number of observations comprising a calibration dataset. The way in which DSI1 accomplishes this task is now described.

If $\mathbf{C}_{22}$ is subjected to singular value decomposition, we obtain

$$\mathbf{C}_{22} = \mathbf{U}\mathbf{S}^2\mathbf{U}^t \quad (18.1.6)$$

(**S** is squared in the above equation because of its relationship to a matrix that will be discussed below.) A generalized inverse of $\mathbf{C}_{22}$ is then obtained as

$$\mathbf{C}^{-1}_{22} = \mathbf{U}\mathbf{S}^{-2}\mathbf{U}^t \quad (18.1.7)$$

When using equation 18.1.7 to construct $\mathbf{C}^{-1}_{22}$, truncation takes place where singular values are zero or near-zero. This cannot exceed the number of realizations that are used for

prediction conditioning. Nor can it exceed the number of observations that are used for prediction conditioning.

Let us label the matrix of noise-supplemented model outputs minus their mean, divided by the square root of (*NREAL*-1) as **Z**. That is

$$\mathbf{Z} = \frac{(\mathbf{X}_2 - \widehat{\mathbf{X}}_2)}{\sqrt{NREAL-1}} \tag{18.1.8}$$

In equation 18.1.8, $\mathbf{X}_{2ij}$ is the *j*'th realization of the noise-supplemented *i*'th model output used in the calibration process. $\widehat{\mathbf{X}}_2$ is a matrix in which the $\boldsymbol{\mu}_2$ column vector is repeated NREAL times; $\boldsymbol{\mu}_2$ is the vector of means of calibration-pertinent, noise-supplemented model outputs. All of **Z**, $\mathbf{X}_2$ and $\widehat{\mathbf{X}}_2$ have as many rows as there are observations used in the history-matching process, and as many columns as there are realizations; that is, they are all NOBS×NREAL matrices. For model output *i*, the mean μ_{2i} of these outputs over all realizations is computed as

$$\mu_{2i} = \frac{\sum_j^{NREAL} \mathbf{x}_{2ij}}{NREAL} \tag{18.1.9}$$

$\mathbf{C}_{22}$ can then be computed as

$$\mathbf{C}_{22}=\mathbf{Z}\mathbf{Z}^t \tag{18.1.10}$$

(Note however that DSI1 does not actually compute this matrix.) If **Z** is subjected to singular value decomposition, we obtain

$$\mathbf{Z} = \mathbf{U}\mathbf{S}\mathbf{V}^t \tag{18.1.11}$$

where **U** and **S** featured in this equation are the same as those featured in equations 18.1.6 and 18.1.7. However instead of undertaking singular value decomposition of the NOBS×NREAL matrix **Z** (which is generally cheaper than undertaking singular value decomposition of the NOBS×NOBS matrix $\mathbf{C}_{22}$), DSI1 undertakes singular value decomposition of the NREAL×NREAL matrix $\mathbf{Z}^t\mathbf{Z}$. Decomposition of this matrix leads to:

$$\mathbf{Z}^t\mathbf{Z} = \mathbf{V}\mathbf{S}^2\mathbf{V}^t \tag{18.1.12}$$

Where **V** and **S** in this equation are the same as those featured in equation 18.1.11. From this equation:

$$\mathbf{U}\mathbf{S}^{-1} = \mathbf{Z}\mathbf{V}\mathbf{S}^{-2} = \mathbf{G} \tag{18.1.13}$$

where **G** is defined through the above equation. Hence

$$\mathbf{C}^{-1}{}_{22} = \mathbf{G}\mathbf{G}^t \tag{18.1.14}$$

With appropriate singular value truncation, $\mathbf{C}^{-1}{}_{22}$ calculated using equation 18.1.14 is the same generalized (i.e. low rank) inverse as that calculated using equation 18.1.7. However the numerical cost of obtaining this low rank inverse approximation is relatively small.

18.1.2 Some Usage Details

18.1.2.1 PEST Control File

DSI1 reads a PEST control file. It obtains the measured values of observations, as well as the weights associated with observations, from this file. DSI1 assumes that the standard deviation of noise associated with each measurement comprising the calibration dataset is equal to the inverse of the weight associated with that measurement; it also assumes that measurement noise is uncorrelated between measurements. Observations to which weights of zero are

ascribed in a PEST control file are not used for conditioning of a prediction.

DSI1 does not read any other part of a PEST control file other than the "observation data" section of that file. Prior to undertaking model runs that are used for the construction of the $\mathbf{C}_{22}$ and $\mathbf{C}_{12}$ matrices discussed above, a user must generate random samples of the prior parameter probability distribution. However it is not necessary for the parameters which comprise these samples to be the same as those represented in a PEST control file; they can be geostatistical realizations of a far more complex stochastic field. DSI1 takes no interest in the number and nature of parameters featured in a PEST control file, nor in their prior probability distribution, when reading this file.

Because DSI1 reads only the "observation data" section of a PEST control file, the file supplied to it by a user does not actually need to be a PEST control file at all. Any file that contains a section which begins with an "* observation data" header can be supplied. This section can be terminated by any line that begins with "*", or by the end of the file. Comment lines (which begin with "#") and lines that contain PEST++ control variables (which begin with "++") are ignored.

18.1.2.2 CSV Files

Prior to employing DSI1, a user must run a model many times using different realizations of parameter values. DSI1 looks for the outcomes of these model runs in CSV files. These CSV files can be produced by members of the PEST++ suite, including PESTPP-SWP and PESTPP-IES. (Such files can also be filled using the contents of run results files written by PEST_HP after translation to CSV format using the RRF2CSV utility.) DSI1 expects the first row of a CSV file that it reads to contain observation names (except for the first column which generally contains the character string "real_name"). Each subsequent row should contain a realisation name (generally an integer or the word "base"), followed by the values of model-generated observations.

As will be described, DSI1 prompts for two CSV files. One of these should hold realisations of all observations that are featured in the PEST control file that it reads. This is referred to herein as the "calibration CSV file". Observations should be listed in the same order as in the PEST control file. Many realisations of observation values should be provided – one on each row of the CSV file following the header. Hence this file could have been written by PESTPP-SWP or PESTPP-IES when run under the control of this same PEST control file.

The other CSV file which is read by DSI1 is referred to herein as the "predictive CSV file". The contents of this file are less restrictive than those of the calibration CSV file. DSI1 requires only that one of its columns is devoted to the prediction whose uncertainty is being examined; it ignores other columns. This file must record values for this prediction that correspond to the realisations of model outputs that appear in the calibration CSV file. The ordering of these realisations must be the same in both files. DSI1 does not cross-check realisation names (i.e. row names) between these CSV file in case realisations are inadvertently misnamed. It is up to the user to ensure that model outputs which comprise a particular row of the calibration CSV file were computed using the same parameter set as those which comprise the same row of the prediction CSV file.

Note that there is no reason why the calibration CSV file and the prediction CSV file cannot be the same. This occurs when a prediction is "carried" in the calibration dataset, presumably with a weight of zero.

18.1.2.3 Model Output Transformations

The theory on which the DSI1 algorithm is based is correct only if model outputs under both calibration and predictive conditions, as well as the noise associated with measurements comprising a calibration dataset, have Gaussian distributions. Rarely will this be the case in practice. To accommodate this (at least to some extent) DSI1 allows model outputs (as well as observations to which they correspond) to undergo primitive transformation before formulation of the $\mathbf{C}_{22}$ and $\mathbf{C}_{12}$ matrices discussed above. Optionally, the prediction can be similarly transformed. Note however that different transformations can be assigned to different model outputs, and to the prediction.

DSI1 prompts for the name of an "observation transformation file". An example of such a file is shown in figure 18.1. (Note that it is not essential that such a file be provided to DSI1.)

```
observation   scale  offset trans baseval
 well01_1       2.0     2.0  1   1e-5
 well02_1       2.0     2.0  1   1e-5
 well03_1       2.0     2.0  1   1e-5
 well04_1       2.0     2.0  1   1e-5
 well05_1       2.0     2.0  1   1e-5
 well06_1       2.0     2.0  1   1e-5
 well07_1       2.0     2.0  1   1e-5
 well08_1       2.0     2.0  1   1e-5
 well09_1       2.0     2.0  1   1e-5
 well10_1       2.0     2.0  1   1e-5
 well11_1       2.0     2.0  1   1e-5
 well12_1       2.0     2.0  1   1e-5
 part_time      0.0     1.0  0   1e-5
 part_east      0.0     1.0  0   1e-5
```

Figure 18.1 Part of an observation transformation file.

The first line shown in the above example of an observation transformation file is optional.

The first column of an observation transformation file must list observations featured in the PEST control file. These must be recorded in the same order as in the PEST control file. Observations featured in a PEST control file cannot be omitted from an observation transformation file even if they have a weight of zero in the PEST control file.

The next two columns of an observation transformation file must contain values of "scale" and "offset". These are applied to each observation recorded in the PEST control file, as well as to all model output realizations that correspond to that observation (after a realization of measurement noise has been added to each such output). When applying scale and offset, DSI1 first multiplies by the scale, and then adds the offset.

The next column of an observation transformation file must be comprised of integers. Each integer must be either 0 or 1. If its value is 1 for a particular observation, then the value of the observation and the values of corresponding noise-supplemented model outputs are log-transformed after application of the scale and offset. Under these circumstances, a number must fill the fifth column of the observation transformation file. If the value of the observation or a corresponding noise-supplemented model output, after multiplication by the scale and addition of the offset, is less than the value recorded in this column then it is elevated to this number before log transformation. This number is referred to as the "emergency base value"; its value must exceed zero.

Optionally, values for these same transformation variables can be supplied for the prediction whose uncertainty is being analyzed through a particular DSI1 run. However, for a

prediction, the values of these transformation variables are provided in response to pertinent DSI prompts. If a prediction is featured in a PEST control file, then transformation variables associated with this model output in the observation transformation file are ignored if the prediction has a weight of zero (which is normally the case). The values of predictive transformation variables (if any) supplied through screen prompts take precedence when a particular model output fills a predictive role in calculation of the $\mathbf{C}_{12}$ matrix.

18.1.2.4 Gaussian Transformation

Instead of (or as well as) undertaking scale, offset and/or log transformation, model outputs and predictions can be individually subjected to Gaussian transformation; first they are subjected to normal score transformation, and then to transformation to a normal distribution with a mean of zero and a standard deviation of 1.0. This can often be a more effective means of achieving Gaussianality than the transformations discussed above. However problems may be encountered when field observations are incompatible with an assumed description of prior parameter uncertainty. This is further discussed below.

18.1.3 Using DSI1

DSI1 receives its information from user responses to its prompts. However, in contrast to members of the Groundwater Data Utility suite, responding to a prompt with "e" does not instigate back tracking. If enough users find this irritating, this functionality will be added.

DSI1 begins execution with the prompt:

```
Enter seed for random number generator (<Enter> if 1111):
```

DSI1 employs random number generation in order to add realizations of measurement noise to calibration-pertinent model outputs calculated from realizations of parameters. As stated above, realizations of model outputs are read from a CSV file. The standard deviation of noise associated with each measurement is equated to the inverse of the weight associated with that measurement in a PEST control file.

Next DSI1 asks for the name of a PEST control file. The prompt is

```
Enter name of PEST control file:
```

As has already been discussed, the PEST control file which DSI1 reads need not actually be a PEST control file at all. The file which DSI1 reads need only contain the "observation data" section of a PEST control file. DSI1 reads observation names, values and weights from this section. Observations with weights of zero are ignored in calculating posterior predictive uncertainty. However, for the sake of compatibility, their presence is expected in other files read by DSI1 which are linked to this PEST control file. These are the observation transformation file and the calibration CSV file; see below.

Next, DSI1 asks the user whether an observation transformation file should be read. The prompt is:

```
Enter name of transformation file (<Enter> if none):
```

If the response to this prompt is "<Enter>", then DSI1 ascribes a scale of 1.0 and an offset of 0.0 to all observations comprising the calibration dataset and all corresponding, noise-supplemented model outputs; nor are these numbers log-transformed. Alternatively, if a filename is supplied, DSI1 asks:

```
Skip a line at the top of this file? [y/n]:
```

DSI1 then reads the observation transformation file. If observations in this file are not listed in the same order as in the PEST control file, DSI1 will cease execution with an error message. This applies even to observations that are assigned weights of zero in the PEST control file.

DSI1 next asks for the name of the calibration CSV file. The prompt is:

```
Enter name of CSV file containing model-generated observations:
```

Observations must be listed in this file in the same order as in the PEST control file (including zero-weighted observations). This, of course, will be ensured if the PEST control file was used to produce the CSV file (as will often be the case). DSI1 then asks:

```
How many realizations to read from this file?
```

Suppose that the answer to this question is "100". Then DSI1 reads model output values corresponding to the first 100 realizations that it finds in this file, regardless of how many other realizations are also present in the file.

Next DSI1 prompts for the name of a CSV file which contains the prediction of interest (i.e. the prediction CSV file). This may or may not be the same CSV file that it has just read. The prompt is:

```
Enter name of CSV file holding N prediction realizations:
```

where *N* is the number of realizations read from the calibration CSV file. It is important to note that when DSI1 reads the prediction CSV file, it does not check whether the model outputs that it finds in this file correspond to observations listed in the PEST control file; it simply looks for the user-specified prediction. Hence the CSV file can be based on a different PEST control file from that which was used to generate the calibration CSV file; the latter PEST control file is the one read by DSI1. The name of the single model output in which DSI1 is interested when reading this file must be supplied by the user in response to the following prompt:

```
Enter name of prediction:
```

As has already been discussed, it is the user's responsibility to ensure that random parameter sets that were used to calculate random model outputs were the same for both the calibration and prediction CSV files. DSI1 reads as many realizations of the prediction from the prediction CSV file as it previously read realizations of calibration model outputs from the calibration CSV file. It assumes that ordering of realizations is the same in both of these files.

DSI1 next asks if the prediction should be transformed before calculating its covariance with respect to model outputs corresponding to observations comprising the calibration dataset; in some cases, this may render its distribution closer to Gaussian. The prompt is

```
Transform prediction? [y/n]:
```

If the answer is "y" DSI1 then asks:

```
Enter scale to apply to prediction:
Enter offset to apply to prediction:
Log-transform scaled offset prediction?
```

and if the answer to the last of the above questions is "y":

```
Enter emergency base value for scaled offset prediction:
```

The roles of these transformation variables are the same as those of the same name featured in the observation transformation file; see the above description.

DSI1 next asks whether it should undertake covariance calculations in Gaussian transform space. The prompt is

```
Work in Gaussian transform space? [y/n]:
```

A response of "y" to the above prompt often works well. This matter is further discussed below.

DSI1 next prompts for the name of the file to which it must record prior and posterior predictive means and uncertainties. It asks:

```
Enter name for output file:
```

and then:

```
Enter "energy threshold" for SVD truncation.
 (Normally between 0.9 and 0.9999):
```

The amount of "energy" is calculated through summing the singular values of **Z**; see equation (18.1.11). Enough singular values are retained for their sum (relative to the sum of all singular values) to be equal to, or greater than, the "energy threshold" that is supplied in response to the above prompt. The best response is normally "0.99".

18.1.4 DSI1 Output File

An example of a DSI1 output file follows.

```
PROBLEM DETAILS ---->

 Name of PEST control file                       : case.pst
 Name of observation transform file              : none
 Name of observation CSV file                    : case_dsi.0.obs.csv
 Name of predictive CSV file                     : case_dsi.0.obs.csv
 Number of realizations read from these files    : 100
 Name of prediction                              : part_time
 Minimum sampled prediction value                : 3706.990
 Maximum sampled prediction value                : 14523.70
 Prediction transformations                      :
     Scale               = 1.000000
     Offset              = 0.000000
     Log-transform       = no
     Emergency base value = na
 Work in Gaussian transform space                : yes
 Energy level for truncation                     : 0.9999000
 Number of singular values before truncation     : 12


 OUTCOMES OF ANALYSIS ---->


 Predictive confidence limits in Gaussian transform space:-

 distribution      low_95              low_68              mean                high_68             high_95
 ------------      ------              ------              ----                -------             -------
 prior             -1.920882           -0.9633510          -5.8199945E-03      0.9517110           1.909242
 posterior         -1.995859           -1.161842           -0.3278259          0.5061906           1.340207


 Predictive confidence limits in untransformed space:-

 distribution      low_95              low_68              mean                high_68             high_95
 ------------      ------              ------              ----                -------             -------
 prior             3924.373            5171.565            6413.483            7636.669            12087.33
 posterior         3845.423            5056.305            5918.843            7017.086            10536.10


 Prior-data conflict indices of non-zero-weighted observations:-

 observation           min_real_val        mean_real_val       max_real_val        obsval              pdc_index
 -----------           ------------        -------------       ------------        ------              ---------
 well01_1                1.49515000          68.1776683          30.1531000          5.74724007          0.148373839
 well02_1                1.66041000          69.8030242          33.2048000          5.69615984          0.127938750
 well03_1                1.49349000          72.1139100          38.3098000          5.49199009          0.108606759
 well04_1               0.725233000          47.4132519          15.8661000          4.09189987          0.222356281
 well05_1               0.900822000          46.7658877          16.3874000          3.92014003          0.194963537
 well06_1                1.02681000          46.5632675          15.4409000          4.05990982          0.210426036
 well07_1               0.481217000          27.5352605          14.5626000          2.59368992          0.150018852
 well08_1               0.544172000          27.4636448          14.2949000          3.25978994          0.197489031
 well09_1               0.593854000          28.7583046          15.5558000          2.97368002          0.159058589
 well10_1               0.126394000          9.60357642          18.5910000          1.12416995          5.403721833E-02
 well11_1               0.170868000          8.31141883          3.69848000          1.20360994          0.292759505
 well12_1               0.163984000          8.13331983          4.47377000          1.19255996          0.238660564

 Number of observations with pdc_index less than 0.0     = 0
 Number of observations with pdc_index greater than 1.0  = 0
```

Figure 18.2. A DSI1 output file.

The first part of a DSI1 output file echoes user input data. The minimum and maximum (untransformed) values of the prediction as read from the prediction CSV file are also recorded in this section.

The outcomes of predictive uncertainty analysis follow. The limits of the prior and posterior 68% and 95% predictive confidence intervals are tabulated. Where Gaussian transformation is undertaken these outcomes are provided in both standard Gaussian and predictive space. In the latter case, back transformation from scale, offset and/or log-transformation space is also undertaken before listing of confidence limits.

Sometimes a calculated confidence limit and/or a posterior mean may lie outside the range of prior predictive realizations. This may be an outcome of severe model nonlinearity. Alternatively, or as well, the assumption of observation-independent Gaussian measurement noise may be violated because of model structural defects in its simulation of real world behaviour. On other occasions, an unduly high or low posterior mean or confidence limit may indicate inappropriateness of user-characterization of prior parameter uncertainties. For example, the latter may be higher than anticipated, or show different spatial patterns than anticipated. Where Gaussian transformation is undertaken, confidence limits which are outside of the range of model-calculated predictions (as read from the prediction CSV file) cannot be evaluated. Hence the “greater than” or “less than” sign is used to indicate these

extra-range limits. In contrast, where Gaussian transformation is not undertaken, these limits can be calculated, notwithstanding their out-of-range values. However the user should take note of their out-of-range status and draw appropriate conclusions.

The DSI1 output file finishes with a table which lists the "prior-data-conflict index" (i.e. the *pdc_index*) of every observation that is used for conditioning. This index is calculated by subtracting the measured value of an observation from the minimum realized value of the observation and then dividing this by the difference between the maximum and minimum realized values of the same observation. If there is no prior-data conflict, the *pdc_index* lies between zero and one. Values outside of this range indicate prior-data conflict. The number of incidences of prior-data conflict are listed following the table.

18.1.5 Experience and Recommendations

For cases on which performance of DSI1 has been tested so far, the outcomes of its calculations appear to be reasonable. However its performance seems to deteriorate where the number of realizations used for calculation of $\mathbf{C}_{22}$ and $\mathbf{C}_{12}$ is less than about 100.

Conceptually, $\mathbf{C}_{12}$ should be calculated without noise enhancement of model outputs. However $\mathbf{C}_{22}$ should include the effects of measurement noise. This can create problems with implementation of equation (18.1.5), especially where prior predictive uncertainties are high and posterior predictive uncertainties are low, for this requires the subtraction of two large numbers to obtain a smaller one without a mathematical guarantee that the resulting number will be greater than zero. This appears to affect calculations undertaken in Gaussian transformation space more than calculations that are not undertaken in this space. This problem becomes apparent to the user if posterior predictive uncertainty upper and lower bounds are calculated to coincide. Hence there appear to be circumstances where use of a Gaussian transformation should be avoided.

Where prior-data conflict is encountered, the performance of DSI1 may be degraded. This is particularly the case where Gaussian transformation is undertaken. Under these circumstances, DSI1 may not be able to calculate upper and/or lower predictive confidence limits. Instead it may simply indicate that these are below/above the smallest/largest value recorded for the prediction in the CSV file from which it obtained predictive values. This is an outcome of the fact that normal score transformation (undertaken as a precursor to Gaussian transformation) cannot proceed beyond the limits of available data on which to base this transformation.

Where performance of DSI1 is degraded because of an apparent conflict between the prior probability distribution ascribed to parameters on the one hand, and to historical measurements of system state used in the history-matching process on the other hand, then DSI1 may still be profitably used without working in Gaussian transform space.

In some contexts, some simple steps can be taken to avoid the problem of out-of-range predictive confidence limits. Obviously, assignment of higher values to prior parameter variances may ensure that observations fall within model-calculated ranges. Other precautions may be more subtle (but just as effective). For example, if working with a groundwater model, it may be wise to avoid the assumption of a high prior mean permeability value, for this may promulgate low vertical and lateral gradients in model-calculated heads at the same time as it promulgates insensitivity of these heads to parameters. If real-world permeabilities are in fact low, then observed heads may not lie within model-calculated ranges for these heads when prior parameter variances are superimposed on a high prior mean.

18.2 DSI2, DSIMOD and POSTDSIMOD

18.2.1 Background

DSI2 is similar to DSI1 in some respects, but different in many others. It does not perform data space inversion itself. However it creates files that allow this to be easily accomplished using the PESTPP-IES ensemble smoother or PEST_HP.

DSI2 computes an empirical covariance matrix that effectively links the past to the future without involving model parameters. It obtains numbers on which to base this matrix from a CSV file that includes many realisations of model outputs, presumably based on many different model parameter sets. This CSV file may have been written by PESTPP-SWP. Alternatively, it may have been written by PESTPP-IES or by PAR2CSV based on a "run results file" (i.e. an RRF file) produced by PEST_HP. Or it may have been written by another program altogether. Furthermore, in accordance with data space inversion theory, there is no reason for parameter sets on which model runs were based to be samples of a Gaussian probability distribution. Nor is there any need for model outputs to be continuous with respect to stochastic model parameters, for the DSI process does not require parameter adjustment.

DSI2 reads a PEST control file (or simply the "observation data" section of a partial PEST control file), which ascribes values and weights to model outputs of interest. (As is described below, this file can also ascribe measurement noise standard deviations to model outputs of interest.) Model outputs with weights of zero are deemed to be predictions. Those with positive weights are deemed to be observations of system state. Unless a noise standard deviation is specifically assigned to an observation, the observation weight is presumed to be equal to the inverse of the standard deviation of measurement noise that is associated with the corresponding field observation.

Note that DSI2 ignores the "observed" values that are ascribed to model predictions (that is, "observations" with zero-valued weights). Nevertheless, it transfers the entire contents of the "observation data" section of this (partial) PEST control file to a new PEST control file. This file can be used by PEST, PEST_HP or PESTPP-IES for history-matching purposes. None of these programs use zero-weighted model outputs for history-matching.

18.2.2 Theory

The theory on which DSI2/DSIMOD is based is explained by Lima et al (2020) and references cited therein. As is described below, POSTDSIMOD extends the functionality of DSIMOD by accomplishing rapid posterior uncertainty analysis.

As previously stated, DSI2 builds a joint covariance matrix that encapsulates a statistical linkage between all observations and predictions that are cited in a partial PEST control file. Prior to building this matrix, it optionally undertakes basic transformations of these quantities using values of scale, offset and log transformation that are specific to each observation/prediction. The transformed value of each model output is calculated as:

$$m^t = s \times m + o$$

where m and m^t are the native and transformed values of a model output, s is the scale applied to the output and o is the offset applied to the output. These (and log-transformation) can be different for each observation/prediction. Following this transformation, a further transformation to a space in which individual variables are characterized by independent Gaussian distributions can also be undertaken. This transformation is accomplished using

(smoothed) normal scores.

Let the vector **d** depict model outputs that are processed by DSI2. As stated above, some of these model outputs possess field-measured counterparts, while others do not. As also stated above, field measurements and weights, as well as the names of all of these model outputs, are read from the "observation data" section of a PEST control file (or from a file that contains only this part of a PEST control file). Meanwhile, an ensemble of model-calculated values for these model outputs (both observations and predictions) must be read from a CSV file. DSI2 transforms these values according to previously-supplied transformation specifications. Then it calculates an empirical $\underline{\mathbf{d}}$ (the mean of **d**), as well as an empirical C(**d**) (the joint covariance matrix of all elements of **d** over all realizations) from the (transformed) contents of the CSV file. From this it calculates $C^{½}(\mathbf{d})$. Calculation of $C^{½}(\mathbf{d})$ requires that singular value decomposition be performed on the empirical (and possibly rank-deficient) C(**d**) matrix which DSI2 calculates. This is accomplished using a numerically efficient strategy which resembles that which is described in documentation of program DSI1 (see above). The user chooses the "energy" that is retained in $C^{½}(\mathbf{d})$; this determines the number of singular values that are employed in construction of $C^{½}(\mathbf{d})$ from C(**d**). It also determines the number of PCA parameters (i.e. DSI model parameters) that are adjusted in the history-matching process that is described next.

DSI2 stores the $C^{½}(\mathbf{d})$ matrix which it calculates, together with other data, in a binary file that is read by the DSIMOD and POSTDSIMOD programs. DSIMOD is a surrogate model that uses PCA parameters instead of model parameters. "PCA" stands for "principal component analysis". The dimensions of PCA space (i.e. the number of PCA parameters) are determined by the user-specified singular value energy level.

To illustrate how this works, suppose that a random vector **p** is generated in PCA space using a Gaussian distribution whose mean is zero and whose covariance matrix is the identity matrix **I**. Every element of **p** is therefore independent of every other element of **p**; furthermore, each has a standard deviation of unity. Suppose that a vector **f** is calculated from **p** using the equation:

$$\mathbf{f} = C^{½}(\mathbf{d})\mathbf{p}$$

Simple propagation of variance shows that the covariance matrix of **f** is C(**d**). With addition of $\underline{\mathbf{d}}$, this vector comprises a random vector in (transformed) model output space. It can be back-transformed to true model output space by reversing transformations specified by the user when he/she runs DSI2.

18.2.3 Some Algorithmic Details

Tasks performed by DSI2 and DSIMOD are now explained in detail. (Details of POSTDSIMOD are explained in a later subsection.)

DSI2 writes a PEST input dataset. Adjustable parameters pertain to PCA space. Each parameter has a prior mean of zero and a prior standard deviation of 1.0. There may be far fewer of these parameters than there are model outputs that are used in the DSI process. There will almost certainly be far fewer PCA parameters than model parameters. (The model that is used to generate the DSI2-perused CSV file may not even possess "parameters"; random hydraulic properties may have been generated by a geostatistical field generator.) The model that is cited in the DSI2-generated PEST control file is DSIMOD. This is a surrogate model that is history-matched against past measurements of system states and fluxes while predicting future system states and fluxes. DSIMOD calculates surrogate model

outputs (both past and future) from these PCA parameters.

The values of PCA parameters (**p** in the previous section) are provided to DSIMOD by PEST/PEST_HP or PESTPP-IES. DSIMOD then calculates surrogate model outputs (using **d** and $C^{½}$(**d**) provided to it by DSI2). Model outputs with which positive weights are associated are history-matched against observed values; these observed values are cited in the "observation data" section of the partial PEST control file which was read by DSI2, and which DSI2 transfers to a new PEST control file.

As PCA parameters are adjusted by PEST/PEST_HP or by PESTPP-IES, surrogate values of model predictions are also altered. History-matching of surrogate model outputs to their non-zero-weighted field-measured counterparts ensures that surrogate predictions are constrained by field data. If history-matching is undertaken using PESTPP-IES, then the posterior probability distributions of model predictions are thereby sampled. If PEST (or PEST_HP) is used for history-matching instead of PESTPP-IES, then the posterior expected values of model predictions are calculated; POSTDSIMOD can then be used to quantify posterior predictive uncertainty.

18.2.4 Running DSI2

Some of the prompts issued by DSI2 are similar to those issued by DSI1.

DSI2 commences execution with the prompt:

```
Enter name of PEST control file:
```

Supply the name of a PEST control file. Actually, this "PEST control file" need only contain the "observation data" section of a PEST control file. This section must begin with an "* observation data" section header, whether or not it is included in a more complete PEST control file.

The "observation data" section of a normal PEST control file contains four columns of data. The first column contains observation names (20 characters or less in length), the second column contains observation values, the third column contains observation weights, and the fourth column contains observation group names. All of these are transferred directly to the PEST control file that DSI2 writes. As stated above, weights of zero indicate model predictions.

If the DSI2-produced PEST control file is to be used by PESTPP-IES, a user should ensure that the values of non-zero observation weights are the inverse of the standard deviations of noise associated with accompanying measurements. Alternatively, as is described below, measurement noise standard deviations can be supplied directly for the use of PESTPP-IES. These noise standard deviations are also used by POSTDSIMOD; see below. In contrast, if the DSI2-produced PEST control file is to be used by other PEST-suite programs for linear predictive uncertainty and data worth analysis, then weights must be "correct" in the sense described above.

DSI2's next prompt is:

```
Enter name of observation transformation file (<Enter> if none):
```

The format of this file is the same as that described in documentation of program DSI1. Note the following:

- If provided, this transformation file must include all observations/predictions that are cited in the "observation data" section of the (partial) PEST control file that DSI2

reads.

- They must be provided in the same order as that in which they are provided in this (partial) PEST control file.

Next DSI2 asks the following questions:

```
Enter name of CSV file containing model-generated observations:
How many realizations to read from this file?
A realisation per row or per column in this file? [r/c]:
```

As is apparent from the above prompts, two options are allowed for arrangement of data within a CSV file. For the first option, realisations of model outputs are provided one-to-a-line. The first column of the CSV file must contain realization names, while subsequent columns must contain model output values. The first line of the CSV file must contain column headers; for all but the first column, these must be the names of model outputs (i.e. "observation names") that appear in the previously-read (partial) PEST control file. A CSV file which adopts these protocols is produced by members of the PEST++ suite (including PESTPP-SWP and PESTPP-IES).

The second CSV file formatting option is the transpose of the above. For this option, values of a particular model output calculated using many realisations are listed along a single row of the CSV file. Thus all model outputs corresponding to a single realization are listed in a single vertical column. However, the first column of the CSV file must contain the names of model outputs (i.e. "observation names") while the first row of the CSV file must contain realization names. A file such as this can be produced by the RRF2CSV utility following a PEST_HP run.

For either of these CSV file options, it is important to note that observations must be provided in the same order as in the PEST control file that is read by DSI2. If this is not the case, DSI2 will notify you of the problem and then cease execution. The problem can then be easily rectified; CSV files can be re-arranged and/or joined together using a spreadsheet program such as EXCEL, or a Python package such as PANDAS.

DSI2's next prompt is:

```
Transform to Gaussian space? [y/n]:
```

Respond with "y" or "n" as appropriate. A response of "y" will often result in better performance of the data space inversion process. However there are subtleties associated with the transformation process that need to be considered. So if your response to the above prompt is "y", DSI2 then asks:

```
For handling of sample limits:
   to respect                 - enter 0
   if linear extrapolation    - enter 1
   if quadratic extrapolation - enter 2
Enter your choice:
```

The variable that is informed by your response to this prompt is named *extrapmode*. It is also used by program POSTDSIMOD.

If the first option to the above prompt is selected then, for any model output (i.e. observation or prediction) for which samples are recorded in the CSV file whose contents are used to train the DSIMOD surrogate model, the outputs of the surrogate model will never exceed the range of those of the real model as recorded in the CSV file. Selection of option 1 allows linear extrapolation of Gaussian transformation beyond these limits if values ascribed to PCA parameters during history matching of the DSIMOD model require this. Selection of option 2

allows quadratic extrapolation. Quadratic extrapolation is generally superior to linear extrapolation as it takes account of curvature of the transformation. However, regardless of which option is selected, extrapolation of Gaussian-transformed variables beyond the limits of the samples on which Gaussian transformation was based are extremely problematical.

Next DSI2 asks for the "energy threshold". This sets the number of singular values before truncation when building the $C^{½}(\mathbf{d})$ matrix. It also determines the number of PCA parameters that are adjusted during DSI model history-matching. "Energy" refers to the sum of singular values of $C^{½}(\mathbf{d})$. Truncation occurs at the singular value at which the "energy" is the user-specified fraction of the total singular value energy.

```
Enter "energy threshold" for SVD truncation.
 (Normally between 0.9 and 0.9999):
```

DSI2's next-to-final prompt is:

```
Enter filename base for output files:
```

Provide a filename base (without extension). Then DSI2 asks:

```
How many IES realisations do you wish to employ?
```

DSI2 records this request in the PEST control file that it is about to write. Note that there is no need for the number of IES realisations to be the same as the number of realisations that are featured in the CSV file from which DSI2 reads realisations of model outputs.

If transformation to Gaussian space was requested, DSI2 thinks for a while. It then writes the following files:

- a prior-data-conflict file;
- a PEST control file;
- template and instruction files that are cited in this PEST control file;
- a DSIMOD/POSTDSIMOD binary input file;
- a DSIMOD/POSTDSIMOD parameter list file in which parameter values are all zero;
- a prior uncertainty file for PCA parameters. (Prior standard deviations are all 1.0.)

The prior-data-conflict file contains one line for each positively weighted observation that is recorded in the PEST control file. For each of these observation it records:

- the minimum value of the observation contained in the realization CSV file;
- the mean realized value of the observation;
- the maximum realized value of the observation;
- the observed value of the observation;
- a crude "prior-data-conflict" index.

The value of the prior-data conflict index lies between zero and one if the measured value of an observation lies between the minimum and maximum realized values of that observation. It is calculated as (observed value minus minimum realized value) divided by (maximum realized value minus minimum realized value). Its value is negative if the measured value of an observation is less than the minimum realized value. Its value is greater than one if the measured value of an observation is greater than the maximum realized value.

The DSI2-produced PEST control file instructs PEST to run in "estimation" mode. As has already been discussed, its parameters reside in PCA space. All have an initial value of zero and are endowed with wide bounds. The prior uncertainties of these parameters are recorded in a parameter uncertainty file whose extension is "*.unc*". (This is used by PESTPP-IES. It can also be used in linear uncertainty analysis if desired; see below.)

The DSI2-produced PEST control file can be used by PEST, PEST_HP or by PESTPP-IES. IES control variables are included in this file. These provide the following information to PESTPP-IES:

- the number of realisations that it should employ;
- the name of the prior parameter uncertainty file;
- that it must employ a subset size of 5 for Marquardt lambda testing.

Other PESTPP-IES control variables can be added manually if desired.

The command that is recorded in the "model command line" section of the DSI2-produced PEST control file runs the DSIMOD model. As is described below, the command line includes the name of the binary file that DSIMOD must read, the name of the parameter list file that contains values of PCA parameters, and the name of the file in which DSIMOD must record observations/predictions that it calculates from these parameters. DSI2 writes a template file corresponding to the parameter list file, and an instruction file that reads the DSIMOD output file. It also writes an example parameter list file in which PCA parameter values are all 0.0. (These are the prior means of these PCA parameters.) This allows you to run DSIMOD without running PEST if you want to make sure that it works.

As has already been discussed, the "observation data" section of the PEST control file that DSI2 writes reproduces that which it obtained from the (partial) PEST control file that was supplied to it. Hence when history-matching is undertaken based on the PEST control file that DSI2 writes, historical data (identified using positive weights) are matched, while predictions are simply "carried" through the history-matching process. When the history-matching process is complete, these predictions are available for inspection in pertinent PEST/PEST_HP or PESTPP-IES output files. In the former case they comprise posterior expected values of predictions; in the latter case they comprise samples of posterior predictive probability distributions.

18.2.5 Supplying Measurement Noise Standard Deviations for PESTPP-IES

As is explained in documentation of PESTPP-IES, random samples of measurement noise are added to observations before parameter realisations are adjusted. PESTPP-IES can obtain the statistics of measurement noise in either of two ways. The "standard" way is to assume that the weight associated with each observation is the inverse of the standard deviation of measurement noise associated with that observation. Alternatively, a set of observation-specific noise standard deviations can be provided along with observation names, values, weights and groups. The unlinking of weights from noise allows a user to balance the contributions to the total objective function made by different observation groups; as is described elsewhere, this can accommodate the presence of model structural noise. Meanwhile, PESTPP-IES draws samples of measurement noise from the user-specified standard deviation which is ascribed to each measurement.

If a user wishes to provide them, PESTPP-IES requires that measurement noise standard deviations be supplied through one or a number of files that are cited in the "observation data" section of a PEST control file. These external files contain observation names, values, weights, groups and (optionally) standard deviations. DSI2 will write a PEST control file which cites an external observation data file if a user supplies observation standard deviations in the file from which it reads observation data. As stated above, this file can be a fragment of a PEST control file or an entire PEST control file. In either case, DSI2 reads only the "observation data" section of this file.

Under normal circumstances the "observation data" section of a PEST control file contains four columns of data. These data are, in order:

- observation names;
- observation values;
- observation weights;
- observation group names.

Optionally, a user may add a fifth column to this file in which observation standard deviations are listed. If this is done, DSI2 writes two PEST control files instead of one. The first of these PEST control files observes standard PEST control file protocols. Furthermore, it contains no PESTPP-IES control variables. Thus it is meant only for the use of PEST or PEST_HP. The second PEST control file is intended for the use of PESTPP-IES. It informs PESTPP-IES that observation data is supplied externally. DSI2 also writes the external observation data file. This is a CSV file which respects PESTPP-IES expectations. It includes a fifth column of data (labelled "standard_deviation") which provides the standard deviation of measurement noise associated with each observation. Meanwhile, observation weights are the same as that supplied by the user in the original partial PEST control file.

Note that if some observations in the original partial PEST control file are ascribed standard deviations while others are not, then the PESTPP-IES-specific control file is still written. In writing this file, observation noise standard deviations are calculated as the inverse of observation weights for those observations which are missing user-supplied standard deviations.

Where two PEST control files are written, the PEST-only PEST control file is named *case.pst* where *case* is the user-supplied filename base for DSI2-produced files. Meanwhile, the PESTPP-IES-only PEST control file is named *case_sd.pst* where the "sd" suffix denotes inclusion of observation standard deviations in the PESTPP-IES observation dataset. The external observation data file is named *case_obsdat.csv*.

As well as informing DSI2 that it must write a second PEST control file for the use of PESTPP-IES, inclusion of standard deviations in the "observation data" section of a PEST control file serves another purpose. These standard deviations are used by the POSTDSIMOD utility to assess post-history-matching uncertainties of DSI model PCA parameters and of model predictions. These uncertainty assessments are often somewhat "cleaner" than those yielded by PESTPP-IES; at the same time, they are calculated more quickly. Their calculation requires assessment of measurement noise. If a user does not supply standard deviations of measurement noise in the "observation data" section of the PEST control file that forms the basis for DSI model construction, then POSTDSIMOD calculates the standard deviation of measurement noise as the inverse of each measurement weight. However, as is often the case in history-matching, weights may serve other purposes; for example they may be used to balance the contributions of different data types in the overall objective function. Furthermore, even though they may reflect observation credibility, they may do so only in a relative sense. Hence, it is likely that POSTDSIMOD's assessments of parameter and predictive uncertainty will be better if a user supplies measurement noise standard deviations separately from weights in the PEST control file that forms the basis for DSI model construction.

An observation that is assigned a weight of zero is taken to be a prediction. However, this too can be assigned a standard deviation of "measurement noise", which therefore becomes the standard deviation of "predictive noise". If asked to do so, POSTDSIMOD can account for

predictive noise when assessing the uncertainty of a prediction. See below.

18.2.6 Negative Weights

Unlike any other member of the PEST suite, DSI2 will not object if a negative weight is assigned to an observation. This comprises a signal to DSI2 that the observation is to be omitted from the data space inversion process.

In most other cases where a PEST control file is used by PEST or one of its utility programs, assignment of a weight of zero to an observation effectively removes it from the inversion process. The observation is still featured in Jacobian matrices and in PEST reporting of model outputs. However it does not constrain any parameters. This allows it to be used as a prediction, if desired.

If an observation has a weight of zero, then it is effectively a prediction from a data space inversion point of view. However its presence adds to the size of covariance matrices that DSI2 must build, and of the singular value decomposition problem that it must solve. If the item which is assigned a weight of zero is not, in fact, a prediction, this can needlessly increase execution time. It may also increase the possibility of numerical noise being encountered because of the handling of large matrices.

If an observation is given a negative weight, then it is simply ignored by DSI2. This allows a user to bring observations that are cited in a PEST control file in and out of the data space inversion process simply by simply altering weights.

18.2.7 Running DSIMOD

If the command "DSIMOD" is typed at the command-line prompt, DSIMOD informs the user how it should be run. It is run using the command:

```
dsimod infile parfile outfile
```

where:

`infile`	is the name of a binary input file written by DSI2;
`parfile`	is a file containing a list of PCA parameter values, and
`outfile`	is a file containing DSI-calculated values of model outputs.

As has already been discussed, the binary input file informs DSIMOD how it should perform its role as a PCA-based model that calculates surrogate outputs of a much more complex simulator. Information that it obtains from this file includes transformation variables, as well as $\underline{\mathbf{d}}$ and $\mathbf{C}^{½}(\mathbf{d})$.

The parameter list file contains a list of current PCA parameter values. DSIMOD records "model output" values that it calculates from these PCA parameter values in its output file. DSI2 provides names for these DSIMOD input and output files; they all have the same filename base, this being provided by the user when he/she ran DSI2. This filename base is the same as that of the DSI2-generated PEST control file.

18.2.8 Running PESTPP-IES

Once DSI2 has written its PEST input dataset, the entire dataset can be checked using PESTCHEK. Suppose that the DSI2-produced PEST control file is named *case.pst*. Then PESTCHEK can be run using the command:

```
pestchek case
```

Note, however, that if a secondary PESTPP-IES-only PEST control file is written, then PESTCHECK will not approve of this file, for it does not support the use of external observation data files. (However PESTPP-IES will happily read this file.)

PESTPP-IES can then be run using the command:

```
pestpp-ies case
```

or

```
pestpp-ies case_sd
```

depending on whether or not a user specifically provides standard deviations of measurement noise.

If measurement noise standard deviations are "correct", or if only weights are supplied and they are equal to the inverse standard deviations of measurement noise, the objective function should be reduced to a value that is about equal to the number of non-zero-weighted observations. PESTPP-IES should be prevented from reducing the objective function below this. Monitor file *case.phi.actual.csv* to see the iteration number at which the desired value of the objective function is achieved. If necessary, accept results from a few iterations before the last iteration which PESTPP-IES undertakes, or halt PESTPP-IES execution when the objective function is lowered to a suitable value. (Use of a surrogate model in place of a real model can introduce an unknown amount of "structural noise" to the history-matching process. Overfitting occurs where estimated values of PCA parameters are unreasonable. Keep in mind that their prior means are zero while their prior standard deviations are one. "Unreasonable" is assessed according to these metrics.)

Suppose that a suitable objective function value is achieved at iteration *J*. Posterior predictive histograms can be constructed using the contents of file *case.J.obs.csv*. This contains *N* realisations of all predictions, where *N* is the number of user-requested PESTPP-IES realisations.

18.2.9 Calibrating the DSI Model

As an alternative to running PESTPP-IES, a user can run PEST (or PEST_HP) to obtain posterior expected values of all predictions; these should lie somewhere near the centres of their respective posterior uncertainty intervals. First add regularisation to *case.pst* using the ADDREG2 utility. Use a command such as the following:

```
addreg2 case case1 target continue
```

This command instructs ADDREG2 to build a new PEST control file named *case1.pst* from *case.pst*. In this new PEST control file *target* (which should be a real number) is the value of the target measurement objective function. If observation weights are "correct" (that is, if they are equal to the inverse of the standard deviations of measurement noise) then set *target* to a value which is a little lower than the number of non-zero-weighted observations featured in the PEST control file. Then run PEST. PEST will run in "regularisation" mode, with preferred parameter values equal to initial parameter values (which are all zero). Once the model is calibrated, the posterior expected values of predictions can be found in file *case1.rec* or file *case1.res* (or *case1.rei* if execution of PEST was halted before it had decided to terminate its own execution).

The "*continue*" option on the above ADDREG2 command line (supplied as the text string "continue") instructs PEST to continue execution even if the measurement objective function falls below *target*. (The REGCONTINUE variable in the "regularisation" section of the

PEST control file is thereby set to "continue".) PEST will then try to reduce the regularisation objective function to as low a value as it can while maintaining the measurement objective function at *target*. This guarantees that the solution to the inverse problem is of minimum error variance (because parameter values depart from their preferred values to the smallest extent possible). However, stop PEST execution when you feel that the solution is good enough.

Experience demonstrates that history-matching of the DSI surrogate model is best undertaken using PEST_HP with the UPTESTMIN control variable set to 25. (PEST_HP's inversion algorithm is a little stronger than that of PEST.) Optimal preparation for a PEST_HP run can be achieved by running ADDREG2 using the following command:

```
addreg2 case case1 target continue hp
```

See documentation of ADDREG2 for further details. Once the DISMOD model has been calibrated by PEST or PEST_HP, it is easy to obtain a PEST control file in which initial parameter values are optimized parameter values. Run the PARREP utility using a command such as:

```
parrep case1.par case1.pst case_soln.pst
```

If NOPTMAX is set to zero in this file (*case_soln.pst* in the above example), then PEST can be used to run the model once using optimised parameter values. For versions of PEST older than 18.0, use the "*hpstart*" command-line switch when running PEST so that PEST ignores the UPTESTMIN control variable. (If HP control variables reside in a PEST control file, then PEST will complain with an error message. However it will ignore these variables if it is run using the "*hpstart*" command line switch and NOPTMAX is set to zero in the PEST control file. Versions of PEST older than 18.0 do not require a "*hpstart*" command line switch if provided with a DSI PEST control file in which NOPTMAX is set to zero.) So run PEST using a command such as:

```
pest case_soln /hpstart
```

So why would one use PEST_HP to calibrate the DSI model instead of using PESTPP-IES to sample parameter (and hence predictive) posterior probability distributions? The reason is that sometimes DSI model outputs do not look very good after this model has undergone history-matching. In particular, model-generated time series that should be monotonic may exhibit spurious gradient reversals. Sometimes they may transgress physical limits. One cause of this problem is overfitting (to field measurements) of a model that provides only approximate simulation of real world behaviour (i.e. the DSI model). However, by using PEST_HP in "regularisation" mode, a user can insist that the DSI model not be overfit. This is done by using an appropriate target measurement objective function (i.e. PHIMLIM). Because the DSI model can be calibrated in a number of seconds, a modeller has the opportunity to re-calibrate the model while setting PHIMLIM to higher and higher values until post-calibration DSIMOD-model outputs are satisfactory. The optimum level of model-to-measurement misfit is thereby established.

18.2.10 Post-Calibration Uncertainty Analysis using POSTDSIMOD

18.2.10.1 What POSTDSIMOD Does

Once the DSI model has been calibrated using PEST_HP, this model can be subjected to posterior uncertainty analysis.

One method of conducting posterior uncertainty analysis is described in section 18.2.8; this

requires the running of PESTPP-IES. If Gaussian transformation of model outputs has been undertaken (see documentation of DSI2), the DSIMOD model is nonlinear. This is because DSI model outputs must be transformed from Gaussian space to model output space on each occasion that the DSI model is run; this is a nonlinear transformation. Nonlinearity, of course, does not prevent an ensemble-based method such as PESTPP-IES from sampling a posterior parameter probability distribution, for the PESTPP-IES history-matching process is iterative.

It is important to note, however, that if a model is linear, then iterative history-matching is not required. In this case, a linearised form of Bayes equation calculates the same posterior predictive probability distribution as that which is sampled using PESTPP-IES. This is because PESTPP-IES uses these same linear equations during each iteration of its history-matching process.

Because the DSIMOD model is, in fact, linear in Gaussian space, a linearised form of Bayes equation can be employed to calculate posterior parameter and predictive uncertainty intervals that pertain to this space following PEST_HP calibration of the DSIMOD model; see below. Once these Gaussian-space intervals are calculated, they can be back-transformed to model output space. As such, they are the same uncertainty limits that PESTPP-IES attempts to respect. However, they possess fewer "bumps and lumps" than those that are evaluated from PESTPP-IES-generated samples of posterior predictive probability distributions. Note, however, that individual model outputs that probe posterior predictive space are not calculated by POSTDSIMOD, for Bayes-based linear formulae calculate these limits directly, rather than inferring them from samples of a posterior probability distribution.

18.2.10.2 Running POSTDSIMOD

Posterior uncertainty limits can be calculated in the manner described above using POSTDSIMOD. This program is actually an expanded version of DSIMOD. Its input files are identical to those of DSIMOD. It is run using the following command:

```
postdsimod infile parfile outfile [extrapmode] [/p] [/r[N]]
```

where:

`infile`	is a binary input file,
`parfile`	is a text file containing a list of parameters or a parameter value file,
`outfile`	is a text output file,
`extrapmode`	(optional) is 0, 1 or 2,
`[/p]`	(optional) indicates whether predictive uncertainty intervals will include "predictive error",
`[/r]`	(optional) instructs POSTDSIMOD to sample the predictive posterior probability distribution, or (alternatively)
`[/rN]`	instructs POSTDSIMOD to sample the predictive posterior probability distribution using integer *N* as the random number seed.

The *extrapmode* control variable, as well as the "/p" and "/r" command line switches are discussed below.

Note that the ability of POSTDISMOD to evaluate predictive uncertainty intervals does not depend on whether (or not) Gaussian transformation is embodied in the DSI model that was built by DSI2. POSTDSIMOD accommodates whatever option was selected by the user.

The parameter list file that is provided to POSTDSIMOD should contain calibrated PCA parameter values. This will happen automatically if POSTDSIMOD is run after PEST has

been used to run the calibrated model once; see the above example in which *case_soln.pst* is the PEST control file. Inspect the "model command line" section of file *case_soln.pst*; use the same command line to run POSTDSIMOD as that which PEST uses to run DSIMOD. Alternatively, if the name of the parameter list file which appears as the second entry on the POSTDSIMOD command line has an extension of "*.par*", then POSTDSIMOD will assume that this file is a parameter value file instead of a parameter list file. Ideally this file will have been produced by PEST_HP when it calibrated the DSI model, or by PEST when it used the PARREP-generated PEST control file to run the calibrated model. Recall that a parameter value file has the same filename base as a PEST control file, but has an extension of "*.par*".

18.2.10.3 The POSTDSIMOD Output File

The POSTDSIMOD output file contains 9 columns. This file is easily imported into a spreadsheet for inspection and processing. The first column provides the names of all observations and predictions that were featured in the PEST control file on which construction of the DSI model was based. (Recall that this file was read by DSI2; these same observations and predictions are featured in the PEST control file that DSI2 created for history-matching of the surrogate DSI model.) The 4th column of the POSTDSIMOD output file contains DSI model outputs. If the parameter list file (or parameter value file) that POSTDSIMOD was asked to read contains calibrated PCA parameter values, then this column contains DSIMOD-generated posterior expected values of DSI model outputs and predictions; these are the same values that reside in the single column of a DSIMOD output file. The other columns of a POSTDSIMOD output file contain posterior DSI model output uncertainty limits corresponding to -3, -2, -1, 1, 2 and 3 Gaussian-space standard deviations. However, these limits (like DSI model outputs themselves) are back-transformed to model output space if model outputs were subjected to Gaussian or other transformation before history-matching. Posterior confidence limits associated with all DSI model outputs are therefore available. These correspond to Gaussian confidence limits associated with 1, 2 and 3 standard deviations.

If Gaussian transformation of model outputs and observations is undertaken, the POSTDSIMOD output file has a 10th column. This provides the "rank index" of pertinent model outputs. The rank index should vary between 0 and 1. However, a depiction of "< 0" indicates that the posterior mean of a model output is less than the value of any random model output pertaining to that observation or prediction in the set of model outputs from which the DSI statistical model was built. Conversely, a depiction of "> 1" indicates that the predicted value is greater than any corresponding random model output of the same quantity. Values near 0 and 1 indicate that the model output value is very close to the lower or upper extreme of random values for that model output that were used in construction of the DSI statistical model.

18.2.10.4 Bumps and Lumps

Where model outputs are time series, plotting of the outer confidence limits of these time series may still exhibit some unseemly bumps and lumps, particularly if the time series corresponds to a model prediction, and particularly if the calibration dataset holds little information that it pertinent to the prediction. This is unavoidable. It is often an outcome of the fact that Gaussian transformation (if requested) is based on a limited number of samples of model outputs – the same model outputs that DSI2 used to build the DSI model. Recall that evaluation of Gaussian transform characteristics was undertaken by DSI2 when it built the DSI model.

It is also possible that uncertainty limits that are calculated by POSTDSIMOD may transgress the boundaries of physical reality. This can be prevented by specifying that Gaussian back-transformation must respect model output ranges provided to DSI2 in the original CSV file on which construction of the DSI model was based. Set *extrapmode* to 0 on the POSTDSIMOD command line to achieve this. If no value is provided for *extrapmode* on the POSTDSIMOD command line, then DSIMOD inherits the value of 0, 1 or 2 that was provided to DSI2 when it originally built the surrogate DSI model.

18.2.10.5 Some Further Details

If a modeller associates standard deviations as well as weights with the history-matching dataset in the partial PEST control file that DSI2 read before building the surrogate DSI model, then POSTDSIMOD uses these standard deviations in calculation of predictive uncertainties. **This is recommended practice.** Otherwise POSTDSIMOD assumes that standard deviations of measurement noise are equal to the inverse of weights supplied in this partial PEST control file. It is important to note, however, that history-matching (as undertaken by PEST_HP) uses the weights that are recorded in this partial PEST control file in formulation of the measurement objective function. If standard deviations are associated with measurement noise, these are used only by POSTDSIMOD to calculate posterior predictive uncertainties.

For those interested, the formula that POSTDSIMOD uses to calculate the posterior covariance matrix of PCA parameters in Gaussian space is as follows. Let $\mathbf{x}$ denote the vector of surrogate model PCA parameters and let $C(\boldsymbol{\varepsilon})$ denote the covariance matrix of measurement noise, calculated in the manner described above and transformed to Gaussian space. We denote the posterior covariance matrix of $\mathbf{x}$ in this space as $C'(\mathbf{x})$. It is calculated using the standard linear Bayesian formula:

$$C'(\mathbf{x}) = [\mathbf{Z}^T C^{-1}(\boldsymbol{\varepsilon})\mathbf{Z} + C^{-1}(\mathbf{x})]^{-1}$$

where $C(\mathbf{x})$ is the prior covariance matrix of $\mathbf{x}$ and $\mathbf{Z}$ is the calibration Jacobian matrix. $C(\mathbf{x})$ is, of course, the identity matrix $\mathbf{I}$. The posterior uncertainty variance of a model prediction s can then be calculated in this space using the standard formula for propagation of covariance:

$$\sigma'^2_s = \mathbf{y}^T C'(\mathbf{x})\mathbf{y}$$

where $\mathbf{y}$ expresses the sensitivity of a prediction to PCA model parameters in Gaussian space.

18.2.10.6 Predictive Error

The preceding equation expresses predictive uncertainty as it arises from nonuniqueness of model parameterisation and random errors that are associated with field measurements that comprise a calibration dataset. However, the uncertainty of a prediction is often greater than this. It may include “predictive noise”. Predictive noise is similar to “measurement noise”. It can be conceptualized as the error in the “measured value” of a model’s prediction. Given the fact that, in everyday model usage, “measurement noise” includes model structural noise, “predictive noise” therefore includes a lack of ability of a model to exactly replicate system behaviour.

With predictive noise taken into account, the previous equation becomes:

$$\sigma'^2_s = \mathbf{y}^T C'(\mathbf{x})\mathbf{y} + \sigma'^2_p$$

where σ'^2_p is the error variance of predictive error. This is calculated from the standard deviation that a user may associate with a prediction in the PEST control file from which a

DSI model is built.

As is discussed above, the original PEST control file which forms the basis for construction of a DSI model (actually, the "observation data" section of a PEST control file) must include a column of weights. A weight of zero indicates a prediction. Optionally, standard deviations of measurement noise may be associated with some or all observations/predictions. These are used only by POSTDSIMOD; they are not used in formulation of the DSI model, nor in its history-matching. As is discussed in the preceding section, standard deviations that are associated with non-zero-weighted observations are used in POSTDSIMOD estimation of PCA parameter uncertainty, and then of the component of predictive uncertainty that arises from this. If a standard deviation is associated with a zero-weighted observation, then this can also be included in calculation of predictive uncertainty. This is done only if the POSTDSIMOD command line includes a "/p" switch.

If POSTDSIMOD is run with the "/p" switch on its command line, "predictive noise" is added to the uncertainty associated with any predictive model output for which a noise standard deviation is supplied.

18.2.10.7 Sampling the Predictive Posterior

Suppose that you run POSTDSIMOD using the "/r" switch ("r" stands for "random"). Then POSTDSIMOD's output file becomes very simple.

Under these circumstances, the POSTDSIMOD output file is comprised of two columns. The first column contains the names of predictions. The second column contains samples of the posterior probability distribution of those predictions. The following should be noted.

- Only predictions are represented in the POSTDSIMOD output file under these circumstances. These are identified as having zero weight in the "observation data" section of the PEST control file from which the DSI statistical model was built. Observations comprising the history-matching dataset are omitted from this POSTDSIMOD output file.
- If POSTDSIMOD is also run using the "/p" switch, then realisations of predictive noise are included in these predictive samples.

The "/r" switch can be invoked in two ways.

The first way is simply to add "/r" to the end of the POSTDSIMOD command line. In this case POSTDSIMOD obtains its random number seed from the system clock. Random numbers that are recorded on its output file are therefore different on each occasion that POSTDSIMOD is run, even if all of its input files are the same.

Alternatively, an arbitrary integer can follow the "/r" character string. This integer must follow these characters without an intervening space. POSTDSIMOD adopts this integer as the seed for the random number generation process. Samples of posterior predictive probability distributions are thus reproduceable as long as POSTDSIMOD input files are unchanged.

The "/r" switch may be useful when implementing optimisation under uncertainty using DSIVC. The DSIVC workflow is described elsewhere.

18.2.11 Workflow Summary

The steps required to build a DSI model, and to evaluate posterior predictive uncertainties, are now repeated. It is assumed that history-matching is undertaken using PEST_HP. See the

above documentation for use of PESTPP-IES instead.

1. Run the real model NREAL times, where NREAL is the number of parameter realisations. Construct a CSV file of model outputs. These outputs include those that are used for history matching, and those that comprise model predictions of interest.
2. Record real-world observations in a file that mimics the "observation data" section of a PEST control file. (We refer to this type of file as a "partial PEST control file".) Optionally, include measurement noise standard deviations, as well as weights, in this file. (This is recommended practice.)
3. Run DSI2 to build a surrogate model. Experience suggests that you should request a high singular value energy level (e.g. 0.99999). Consider setting *extrapmode* to 1 or 2 in response to the pertinent DSI2 prompt if you request Gauss transformation of model outputs; these *extrapmode* settings generally support better fits between DSI model outputs and field data when calibrating the DSI model than an *extrapmode* setting if 0.
4. Use ADDREG2 to add regularisation to the PEST control file that DSI2 builds. Choose an appropriate value for the target measurement objective function. (You may need to experiment with this.) Specify "continue" and "hp" on the ADDREG2 command line.
5. Calibrate the DSI model using PEST_HP.
6. Use PARREP to build a new PEST control file in which initial PCA parameter values (i.e. DSI model parameter values) are calibrated values. Set NOPTMAX to 0 in this new PEST control file.
7. Run PEST; PEST runs the DSI model once. Inspect and/or plot model outputs using information contained in the run record (i.e. REC) or residuals (i.e. RES) file that PEST writes. If there are too many "bumps and lumps" in these DSI model outputs, then go back to step 4 in order to set a higher value for the PHIMLIM target measurement objective function. Or perhaps return to step 3 and set *extrapmode* to 0 when running DSI2 if DSI model outputs are completely unacceptable.
8. Once the DSI model has been successfully calibrated, and a PEST control file has been built using calibrated parameter values as initial values, and PEST has been invoked to run the calibrated DSI model, run POSTDSIMOD to obtain posterior predictive uncertainty intervals. Set *extrapmode* to 0 when running POSTDSIMOD if outer uncertainty limits are unacceptably bumpy or transgress physical limits.

18.2.12 Other Options

As is stated above, if Gaussian transformation is undertaken, then the DSI model is nonlinear. This does not prevent the undertaking of traditional linear analysis. However the results will be approximate.

To prepare for linear analysis, a Jacobian matrix should be calculated for the calibrated DSI model. Using the example presented above, this requires that NOPTMAX be set to -1 or -2 when running *case_soln.pst*. (Recall that this PARREP-constructed PEST control file contains calibrated DSI model parameter values as initial parameter values.)

Once a Jacobian matrix has been obtained (this takes no time at all), linear analysis can be used to calculate quantities such as the following:

- optimum number of singular values to use (PREDVAR1);
- solution and null space contributions to the uncertainty of a prediction (PREDVAR1);
- a linear approximation to the uncertainty of any prediction (JROW2VEC and

PREDUNC1);
- observation worth (PREDUNC5);
- a posterior PCA-parameter covariance matrix (PREDUNC7).

As an alternative to using these utilities one after another, they can be run sequentially and automatically under the control of GENLINPRED to perform a number of pre-set tasks. For all of the above tasks, the prior covariance matrix of PCA parameters are contained in a file that was written by DSI2; this has an extension of "*.unc*".

The posterior PCA parameter covariance matrix can be sampled using RANDPAR, RANDPAR1, RANDPAR2, RANDPAR3 or RANDPAR4. The DSIMOD model can then be run using all of these samples (using PEST_HP, PESTPP-SWP or PESTPP-IES). This will enable sampling of a linear approximation to the posterior probability distribution of any desired model prediction.

PEST's nonlinear predictive analyser (invoked when PEST is run in "predictive analysis" mode) can be used to maximize/minimize the value of a prediction of interest subject to history-matching constraints. Setup for this operation is facilitated using the ADDPRED1 utility. This utility is discussed below.

18.3 DSI3

18.3.1 General

DSI3 performs an almost identical role to that of DSI2, and can be used interchangeably with DSI2. However it differs from DSI2 in two ways. These are as follows.

1. It forms the full covariance matrix between observations and predictions rather than using the shortcut that is described in documentation of DSI1. It then takes the square root of this matrix to form the DSI surrogate model. Under some circumstances, execution of DSI3 may therefore take longer than execution of DSI2.
2. DSI3 issues one extra prompt. If the answer to this prompt is "n" (the default), then the outcomes of a DSI3 run will be almost identical to those of a DSI2 run. However, they will not be completely identical because different numerical steps that are taken in setting up the DSI statistical model will result in slightly different DSI model outputs.

18.3.2 The Extra Prompt

DSI3 introduces modifications to DSI2 which are designed to handle modelling contexts in which the ranges of one or more model predictions are bounded. Suppose, for example, that a prediction of interest is the number of particles that enter a particular model boundary condition. Depending on the values of a particular set of stochastic model parameters, this prediction may have a lower bound of zero and an upper bound of the number of particles that are actually released. Obviously, it is not possible for a boundary-emergent particle count to exceed these bounds.

Suppose now that a DSI surrogate model is being trained. Random sets of parameters are therefore generated, and the model is run using each of these parameter sets. Suppose that the predictive histogram of particle counts at a model boundary of interest looks like the following.

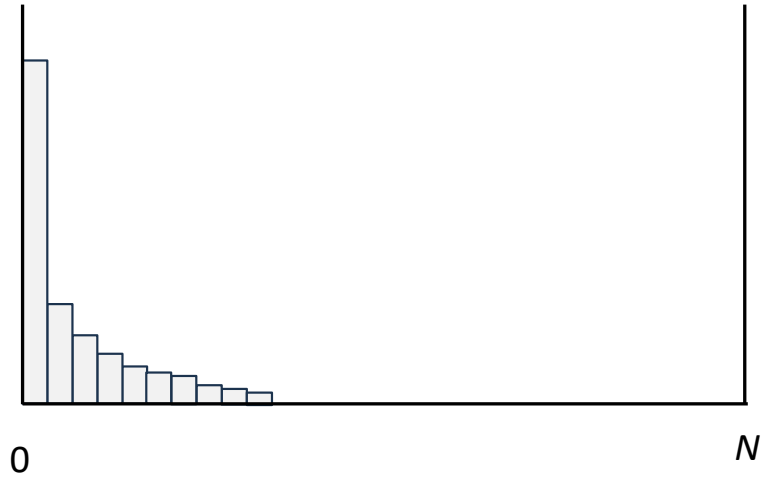


Figure 18.3 A predictive histogram of particle count.

The large bar at the left of this figure is caused by the fact that for many sets of model parameters, zero particles enter the boundary condition of interest. Presumably they emerge at other boundaries (or at extraction wells).

This is an awkward predictive probability distribution, as it does not really obey the rules under which the data space inversion process operates. Proper operation of the DSI process requires that the probability distribution of a model output that corresponds to an observation or a prediction be Gaussian, or is readily transformed to be Gaussian; the latter is possible as long as a certain model output value is not repeated, as occurs in the above example case.

Limited experience to date suggests that the DSI process can better accommodate a situation such as that depicted in Figure 18.3 if calculation of the covariance between the bounded prediction and model outputs corresponding to observations ignores spikes that correspond to either end of a bounded predictive interval. In the above example, incorporation of the left spike limits calculated predictive variance to the extent that it is difficult for the DSI process to make predictions which exceed the lower bound by much, even though predictive values can, in fact, exceed this bound considerably. However calculation of the mean of the prediction should still include predictive samples that are at the lower bound in order to reflect a propensity for predictions to coincide with this bound. These steps are taken by DSI3 if a user's response to its following prompt is "y".

```
Adapt covar calcs for bounded predictions? [y/n <Enter> "n"]:
```

All other prompts issued by DSI3 are the same as those issued by DSI2.

18.4 DSICSVCLIP

DSICSVCLIP addresses the problem that is illustrated in Figure 18.3 but in a slightly different way from program DSI3. Experience to date suggests that use of DSICSVCLIP is effective in overcoming this problem, while being easy to use.

Operation of DSI2 requires the presence of a CSV file containing realizations of model outputs that correspond to field observations and predictions. This CSV file must complement another file whose contents are the same as those of the "observation data" section of a PEST control file. These files may contain one or many predictions. Suppose that you are interested in one particular prediction (which may be representative of others that are in its vicinity). Suppose that realizations of this prediction are limited by an upper and/or a

lower bound so that the prior predictive probability distribution resembles that shown in Figure 18.3.

DSICSVCLIP creates a new CSV file in which realizations are removed from the original CSV file. First the user identifies a prediction of interest. Then he/she informs DSICSVCLIP of threshold values. Realizations are removed from the CSV file if the value that they provide for the prediction of interest is below a user-specified lower threshold or above a user-specified upper threshold. These thresholds should be just inside the bounds of the predictive interval; actually they should be far enough inside these bounds for their values to be correlated with observation values. The new CSV file can then be used in construction of a DSI statistical model. Obviously, this CSV file will contain fewer realizations than the original CSV file. However the statistical model that is built from it will not be compromised by having to accommodate repeated predictive values at the upper and/or lower bounds of a predictive interval, thereby rendering them insensitive to observation values.

Obviously, the CSV file that is built by DSICSVCLIP is useful for the making of one particular prediction. It may also be suitable for the making of other predictions that are similar to this. However there may be other predictions for which DSICSVCLIP should be run again in order to tune the DSI statistical model to the making of that particular prediction.

DSICSVCLIP prompts and typical responses are shown below.

```
Enter name of DSI-ready CSV file: for_dsi.csv
A realisation per row or per column in this file? [r/c]: r

Enter name of corresponding partial PEST control file: observ.pst

Enter name of prediction of interest: pred_12320
Enter lower predictive cutoff: 15.05
Enter upper predictive cutoff: 16.7

Enter name for new CSV file: for_dsi_new.csv
- file observ.pst read ok.
- 200 realizations read from file for_dsi.csv.
- 184 realizations written to file for_dsi_new.csv.
```

If a prediction is not bounded from below, then enter a very low value for the lower predictive cutoff threshold. Likewise, if it is not bounded from above, enter a very high value for the upper predictive cutoff threshold. Obviously, a cutoff threshold has no effect if it is above or below the range of possible predictive outputs.

18.5 ADDPRED1

18.5.1 Background

ADDPRED1 reads an existing PEST control file and writes a new one. The new PEST control file instructs PEST to run in "predictive analysis" mode. It also includes a "predictive analysis" section. As well as this, ADDPRED1 adds a series of prior information equations to the PEST control file that it writes. Each of these equations specifies that the "observed" value of each parameter (or its log), is the initial value of that parameter (or its log).

When run in "predictive analysis" mode, PEST maximizes or minimizes a prediction, subject to the constraint that the objective function rises no higher than a user-specified "limiting objective function"; this is the PD0 variable that appears in the "predictive analysis" section of a PEST control file. It is assumed that the prediction which is to be maximized or

minimized is already listed as an "observation" in the existing PEST control file. (This will probably be the case if that file was written by DSI2.) ADDPRED1 assigns this observation to an observation group named "predict". But first it checks that the weight assigned to this observation in the existing PEST control file is zero. If this is not the case, it will cease execution with an error message.

18.5.2 Running ADDPRED1

ADDPRED1 is run using the following command.

```
addpred1 case1 case2 predname pd0
```

where

case1	is the filename base or full name of an existing PEST control file;
case2	is the filename base or full name of the PEST control file which ADDPRED1 must write;
predname	is the name of an observation in the existing PEST control file; and
pd0	is the limiting objective function that will be recorded in the "predictive analysis" section of the new PEST control file.

ADDPRED1 will object with an error message if the existing PEST control file:

- instructs PEST to run in "predictive analysis" mode;
- contains prior information;
- assigns a non-zero weight to the *predname* observation.

When writing the new PEST control file, ADDPRED1 assigns a value of 1 to the NPREDMAXMIN control variable in the "predictive analysis" section of this file. Hence PEST is asked to maximize the user-nominated prediction. Edit the PEST control file and assign a value of -1 to NPREDMAXMIN if you want PEST to minimize this prediction instead.

18.5.3 Prior Information Equations

ADDPRED1 is similar to ADDREG1 in that it writes one prior information equation for each adjustable parameter that is featured in the "parameter data" section of the existing PEST control file. This equation provides an "observed" value for each parameter that is equal to its initial value (or the log of the parameter's initial value if the parameter is log-transformed in the "parameter data" section of the PEST control file). It follows that the initial contribution to the objective function from prior information is zero.

Each prior information equation is assigned to an observation group whose name is the same as the parameter group to which the respective parameter belongs. Each prior information equation is given a weight of 1.0. Where parameters are DSI PCA parameters, this is the ideal weight for each such equation, for it is equal to the inverse of the prior standard deviation of each PCA parameter.

In most modelling circumstances, predictive analysis is undertaken after model calibration. Therefore the parameters that are featured in the PEST control file that is read by ADDPRED1 should be calibration-estimated parameter values. These may have been emplaced in that file using the PARREP utility.

18.5.4 The Value of PD0

Ideally:

- The weight assigned to each observation should be equal to the inverse of the standard deviation of measurement noise associated with that observation.
- The weight assigned to each prior information equation that expresses a preferred parameter value should be the inverse of the prior standard deviation of the respective parameter.

If observation weights are "correct" in the above sense, then the objective function that arises from observations alone should be roughly equal to the number of observations. "Correct" observation weights can be back-calculated from the measurement objective function obtained during the calibration process using the PWTADJ2 utility.

Chapter 8 of Part 1 of this manual discusses the appropriate setting for PD0. PD0 identifies the objective function above which a model can be deemed to be uncalibrated at a certain level of confidence. If all weights (i.e. measurement weights and weights assigned to prior information equations) are correct, the measurement objective function attained through the calibration process should be roughly equal to the number of observations. If the two-sided 68% confidence limit of a prediction is desired, then add 1.0 to this value to obtain PD0; add 4.0 to this value to obtain its 95 percent confidence limit, and add 9.0 to this value to obtain its 99.7 confidence limit. (These correspond to the square of the number of standard deviations of a Normal distribution. These numbers are approximate; see Chapter 8 of Part 1 of this manual for more information.) This objective limit is imposed on the sum of that incurred by observations and that incurred by prior information equations as the prediction of interest is maximized or minimized. (Recall that the prior information contribution to the total objective function starts out at zero.)

18.6 PDC1

18.6.1 General

PDC1 analyses prior data conflict. Although analysis of prior data conflict is not restricted to use of data space inversion (DSI), it is something that is worth considering when undertaking DSI, as good DSI performance relies on the absence of prior data conflict. Hence program PDC1 is documented in this section of the PEST manual.

DSI2 and related programs report the existence of prior data conflict. However, analyses that are undertaken by these programs are simplistic. Prior data conflict is deemed to occur when one or more measurements are outside the range of model-generated counterparts to these measurements calculated using random hydraulic property fields. This is certainly an indicator of prior data conflict. However prior data conflict can be more nuanced than this. A model may fail to reproduce temporal and spatial *patterns* of field observations, even though the values of its corresponding outputs span observed values of system behaviour when the model is endowed with many stochastic hydraulic property fields. If this is the case, then the prior is incompatible with real-world observations of system behaviour. This indicates a problem with the model, and/or with the nature and patterns of hydraulic property heterogeneity that are embodied in realizations of hydraulic properties with which the model is endowed.

Greater sophistication in analysis of prior data conflict can be gained by implementing a methodology that is described by Alfonzo and Oliver (2019). This methodology is implemented by program PDC1.

18.6.2 Some Theory

Refer to Alfonzo and Oliver (2019) for a full description of the theory behind the methodology that is implemented by PDC1. For ease or reference, the following description employs some of the same symbology that is used in that paper.

PDC1 uses Mahalanobis distance as a basis for analysis of prior data conflict. Let **x** be an *n*-dimensional random vector whose statistical properties are described by its mean $\underline{\mathbf{x}}$ and covariance matrix C(**x**). We now ask whether the vector **z** may be a sample from this same probability distribution. So we calculate the "distance" between **z** and the mean of **x**. However, calculation of this distance takes into account the statistical characteristics of **x** as it employs the covariance matrix of **x**, i.e. C(**x**). Mahanalobis distance squared is calculated using the formula:

$$d_m^2 = D_m = \left(\mathbf{z} - \underline{\mathbf{x}}\right)^T \mathrm{C}^{-1}(\mathbf{x})\left(\mathbf{z} - \underline{\mathbf{x}}\right) \tag{18.6.1}$$

It is easily shown that if $\mathbf{z} \sim N(\underline{\mathbf{x}}, \mathrm{C}(\mathbf{x}))$ (i.e. if **z** shares the same probability distribution as **x**, and if that probability distribution is multiGaussian) then D_m possesses a chi squared distribution with the same number of degrees of freedom as the number of elements of **x** (and hence **z**), i.e. n. A chi squared distribution has a mean of n and a standard deviation of sqrt($2n$). Hence Mahalanobis distance can provide the basis for testing whether **z** may (or may not) be a sample of the same probability distribution as **x**. In doing so, it takes the spatial and temporal characteristics of **z** and **x** into account; the latter are embodied in C(**x**).

In prior data conflict analysis, **z** encapsulates a dataset comprised of field measurements of system behaviour, while **x** represents measurement-noise-contaminated model-generated counterparts to these measurements.

Before the test embodied in equation 18.6.1 can be applied to **z**, $\underline{\mathbf{x}}$ and C(**x**) must be determined. This can be done empirically using stochastic realizations of **x**. For measurement pertinent model outputs these realizations can be generated by running the model using random realizations of model parameters. However, if the number of elements of **x** (i.e. n) exceeds the number of realizations used for empirical evaluation of C(**x**) (we will denote the latter as n_e), this presents us with a problem. In groundwater modelling practice, this problem will be encountered often as the number of observations that are used for model-history-matching against a field measurement dataset often exceeds the number of realizations that are used to generate samples of **x**. In this case, the empirical C(**x**) (we denote it as $\mathrm{C_e}(\mathbf{x})$) is rank-deficient and therefore does not have an inverse. Hence another matrix must be inverted in its place. Ideally this matrix is "close to" $\mathrm{C_e}(\mathbf{x})$ but is not singular. Alfonzo and Oliver (2019) describe such a matrix (see below). PDC1 uses this same matrix.

The Alfonzo and Oliver methodology for prior data conflict analysis (i.e. PDC analysis) proceeds as follows.

First, run the numerical model n_e times based on n_e random realizations of model hydraulic properties. This produces n_e realizations of model outputs **x** encapsulated in the n_e columns of an $n \times n_e$ matrix **X**. Then add realizations of measurement noise to these columns so that these columns can then be considered as model-generated counterparts to actual field measurements.

Next a set of "leave-one-out" calculations is repeated n_e times. During each of these procedures, one member of the **x** ensemble (i.e. one column of **X**) is selected for individual attention. This becomes the "left-out" realization; we denote this realization using the index i. The squared Mahalanobis distance D_{mxi} between this realization and the mean of the

remaining realizations is then calculated. Then the squared Mahalanobis distance D_{mzi} between the set of field measurements **z** and this same mean is calculated.

Specifically:

1. Remove column *i* from **X**.
2. Calculate $\underline{\mathbf{x}}$ and $C_e(\mathbf{x})$ using the remaining n_e-1 realizations.
3. Using equation 18.6.1, calculate the squared Mahalanobis distance between the omitted **x** (i.e. $\mathbf{x}_i$) and the empirical mean of the included realizations. Denote this as D_{mxi}.
4. Using equation 18.6.1 calculate the squared Mahalanobis distance between the measurement dataset **z** and the empirical mean of the included realizations. Denote this as D_{mzi}.

Calculation of $C^{-1}{}_e(\mathbf{x})$ (i.e. the approximate inverse of $C_e(\mathbf{x})$) is described by Alfonzo and Oliver. Exactly the same procedure is used by PDC1 except for the role of the average variance υ that is used in calculation of a matrix **T** that is used to build a new matrix $\widehat{\boldsymbol{\Sigma}}$ to replace $C_e(\mathbf{x})$ using the equation:

$$\widehat{\boldsymbol{\Sigma}} = \delta\mathbf{T} + (1-\delta)\mathrm{C_e}(\mathbf{x}) \tag{18.6.2}$$

Alfonzo and Oliver use a single variance that is averaged over all observations. PDC1 calculates an average variance for each observation group. In contrast to Alfonzo and Oliver, the matrix **T** is therefore no longer the identity matrix **I** divided by υ. It is still a diagonal matrix, with one element pertaining to each observation. However every diagonal element of **T** is the reciprocal of the average variance pertaining to the observation group to which the element belongs. Meanwhile $C_e(\mathbf{x})$ is evaluated using the equation:

$$\mathrm{C_e}(\mathbf{x}) = \frac{\mathbf{X}\mathbf{X}^{\mathrm{T}}}{n_e - 1} \tag{18.6.3}$$

The outcome of the above sequential, leave-one-out procedure is a set of n_e values of D_{mxi} and corresponding values of D_{mzi}. Ideally, values of D_{mxi} obtained in this way provide a basis for deciding whether values for D_{mzi} obtained in this way are different enough to warrant the measurement dataset **z** being identified as statistically different from model-generated datasets **x** contained in **X**. Different statistical characterization of D_{mxi} and D_{mzi} implies prior data conflict.

Following this succession of leave-one-out iterations, the median of the n_e values of D_{mxi} is calculated. This is compared with the median of the n_e values of D_{mzi}. (Note that diversity of D_{mzi} values calculated over the n_e leave-one-out iterations is generally much smaller than that of D_{mxi} values, as there is only one measurement dataset **z** and many incidences of **x**.) A robust approximation to the standard deviation of these n_e D_{mxi} values is then evaluated; this is calculated as 1.4825 times the mean absolute deviation (MAD) from the median of these values. A z-score is then awarded to the median of all D_{mzi} values.

A z-score for a general random variable *r* can be calculated from its mean μ and standard deviation σ as:

$$Z = \frac{r-\mu}{\sigma} \tag{18.6.4}$$

The z-score characterizes how many standard deviations a random variable is removed from its mean. Obviously, high absolute values are unlikely values. For a Gaussian distribution, the interval that is spanned by 2 standard deviations either side of the mean is the 95% confidence interval. A z-score greater than 2 therefore indicates an unlikely value for a

random variable; a z-score greater than 3 indicates a very unlikely value for a random variable. For this reason, a D_{mzi} z-score of 3 or higher, calculated in the above manner, is indicative of prior data conflict.

Alfonzo and Oliver provide another mechanism for appraisal of prior data conflict. They provide a count of how many D_{mzi} values fall within the empirical 95% confidence interval of D_{mxi} values; both of these D_m values are accumulated over the n_e-repeated leave-one-out procedure described above. This count is divided by n_e to yield a fractional count. The closer is this fractional count to 1.0, the less likely is prior data conflict. Program PDC1 reports this fractional value. It also provides information that allows plotting of empirical D_{mxi} and D_{mzi} cumulative probability distributions in order to enable production of graphs similar to those that appear in Figures 5, 6 and 7 of Alfonzo and Oliver.

18.6.3 Using PDC1

Use of PDC1 has much in common with that of DSI2; its reading of input files relies on the same source code. Hence an input dataset for DSI2 can also serve as an input dataset for PDC1. This facilitates prior data conflict analysis (i.e. PDC analysis) as an adjunct to data space inversion.

PDC1 commences execution with the following prompt:

```
Enter name of (partial) PEST control file:
```

Just as for DSI2, this "partial PEST control file" can include nothing more than the "observation data" section of a PEST control file. In fact, this is the only part of the file that PDC1 reads. Just as for DSI2, the partial PEST control file can include an extra column, this containing observation standard deviations.

In a partial PEST control file that is read by DSI2, weights can be positive, zero or negative. Assignment of a negative weight to an observation informs DSI2 that the observation should be ignored. Meanwhile, a weight of zero informs DSI2 that the observation is, in fact, a prediction. Like DSI2, PDC1 ignores observations with negative weights. It also ignores observations with weights of zero, for predictions cannot be included in PDC analysis.

Next PDC1 asks the user to select which observation groups will be used in PDC analysis. As is discussed below (and is also discussed by Alfonzo and Oliver), PDC analysis can often yield more useful results if undertaken sequentially on subsets of a complete history-matching dataset rather than on the entirety of this dataset. PDC1 prompts are as follows ("obsgroup" is just a placeholder name in these prompts):

```
Indicate which observation groups to include in analysis:
  Observation group "obsroup1" [y/n]?
  Observation group "obsroup2" [y/n]?
  Observation group "obsroup3" [y/n]?
  etc.
```

If desired, measurements and corresponding model outputs can undergo transformation before undergoing PDC analysis. As described in documentation of DSI2, PDC1 can read a "transformation file" containing four data columns. Optionally, these columns may be headed by a line that depicts the contents of these columns. The first column must contain observation names, the second and third columns must contain values for scale and offset respectively, while the fourth column must contain the number 1 or 0 signifying whether (or not) observations and their model-generated counterparts should undergo log (to base 10) transformation; "1" requests log-transformation. Transformations can be different for

different observations. First the observation is multiplied by the scale, and then the offset is added. If requested, the resulting number is log-transformed. It is important to note that all observations that are cited in the partial PEST control file that is read by PDC1 must also be cited in the transformation file. Furthermore, they must be cited in the same order. If this is not done, PDC1 will object with an error message. (Note that a scale of one, an offset of zero and a log-transformation index of 0 implies no transformation at all.)

Pertinent PDC1 prompts are:

```
Enter name of observation transformation file (<Enter> if none):
```

Then, if the name of a transformation file is provided:

```
Skip a line at the top of this file?  [y/n]:
```

A response of "y" to the above prompt allows PDC1 to ignore a line containing headers at the top of the file.

Next PDC1 prompts for the name of a CSV file that contains model-generated counterparts to observations cited in the partial PEST control file:

```
Enter name of CSV file containing model-generated observations:
```

As for DSI2, two options are allowed for this file. In the first option, one realization occupies each line of the file following the header line. The ordering of observations along this line must be the same as in the partial PEST control file. Observations must be named in the first row of the file, starting at the second column. All observations must be included, regardless of whether their weights are positive, negative or zero in the partial PEST control file. The first column of the file must contain realization names; these are ignored by PDC1.

The second option is to provide realizations in columns rather than rows. Observation names must therefore occupy the first column (starting from the second row); columns starting at two must contain realizations of values for these observations. The first row of the file must contain realization names; these are ignored by PDC1.

Note that, regardless of the protocol that is followed by this CSV file, the file should provide an entry for (row1, column 1) as a placeholder. This entry is ignored by PDC1.

After obtaining the name of the CSV file that it must read, PDC1 asks how many realizations should be read from this file, and how data are arranged in this file.

```
How many realizations to read from this file?
A realisation per row or per column in this file? [r/c]:
```

Finally PDC1 asks for the name of the file that it must write. The prompt is:

```
Enter name for output file:
```

PDC1 then performs its calculations and ceases execution. These calculations may take a while if there are many realizations (and hence the requirement for many leave-one-out analyses) because of the need to invert the approximate empirical covariance matrix prior to calculation of Mahalanobis distances during each leave-one-out iteration. As many iterations are required as there are realizations.

18.6.4 The PDC1 Output File

Information that is recorded in the PDC1 output file may seem a little confusing. So it is explained here.

The first section of the PDC1 output file reports files that are read by PDC1. It does not

record the contents of these files. It also reports the number of observations and realizations that are employed in PDC analysis.

Then follow some results. An example of a "results" section is provided below for an observation dataset comprised of only 28 observations.

```
RESULTS

 Squared Mahalanobis distances (LOU = "leave-one-out") --->
   Median for LOU model realization to LOU model realization mean    : 27.31547
   Median for measurements to LOU model realization mean             : 18.59528
   Mean absolute deviation (MAD) for all LOU model realizations      : 7.139070
   Overall measurement distance z-score                              : -.8238846
   Fraction of measurement distances in LOU 95% credibility interval : 1.000000
```

As is explained above, squared Mahalanobis distances are computed as many times as there are realizations. On each occasion, a realization is omitted from the ensemble of model outputs contained in the CSV file that is read by PDC1. The squared Mahalanobis distance from the omitted realization (including PDC1-added measurement noise) to the mean of all other realizations (which also include PDC1-added measurement noise) is then computed. Because the squared Mahalanobis distance has a chi squared distribution with order equal to the number of observations, the median value of these distances should be about equal to the number of observations. As explained above, the squared Mahalanobis distance from the measurement dataset (read from the partial PEST control file) to the mean of these same realizations is also computed. Its median value over all leave-one-out realizations is also reported on the PDC1 output file.

Next PDC1 reports a measure of diversity of left-out realization squared Mahalanobis distances to the means of included distances over the n_e leave-one-out analyses that it has performed. This is the mean absolute deviation (MAD) of these distances from their median. Alfonzo and Oliver explain that this measure of diversity is used in preference to the standard deviation as it is less prone to contamination by outliers. The first and third items in the above PDC1 report are therefore complementary, as they are the mean and MAD outcomes of n_e leave-one-out analyses.

The fourth of the above entries is the most important one. This provides the z-score of the median of measurement squared Mahalanobis distances; this is the median of D_{mzi} values discussed in the theory section. This is calculated by subtracting the median of D_{mzi} values from the median of D_{mxi} values and dividing by 1.4825 times the MAD of D_{mxi} values. (See the flowchart in Alfonzo and Oliver.) An absolute value greater than 3 is a cause for PDC concern, for this indicates that the measurement dataset is (on average) more distant from model outputs than individual, random model outputs themselves.

The final item in the above results list can be readily obtained from data that is tabulated at the bottom of the PDC1 output file. This table contains four columns. The first column is comprised of numbers that start at one and finish at the number of realizations. The next column converts these to rank. Rank starts at zero and ends at one. Items in the first and second columns are therefore proportional to each other. The third column contains ordered outcomes of leave-one-out analyses. It lists the squared Mahalanobis distance from each left-out realization to the mean of the other realizations for each of the n_e leave-one-out analyses that were performed by PDC1; that is, it records D_{mxi} values ordered from lowest to highest. The final column lists ordered squared Mahalanobis distances from the measurement dataset supplied in the partial PEST control file to these same leave-one-out means; that is, it records D_{mzi} values ordered from lowest to highest. Plots of cumulative probability for the contents of the last two columns are easily obtained after importation of this table into a plotting

program; see Figures 5 to 7 of Alfonzo and Oliver. The contents of this table also allow computation of the fraction of entries in the last column that fall within the 95% confidence interval of entries in the third column. This interval begins at a rank value of 0.025 and ends at a rank value of 0.975. *This number is listed as the last item in the above figure of PDC1 results following the overall measurement distance z score.* Obviously, an inclusion fraction of 1.0 indicates very little likelihood of prior data conflict. See Alfonzo and Oliver for more details.

18.6.5 Some Notes and Reminders

Weights and standard deviations

As discussed above, PDC1 reads a CSV file containing the outcomes of many model runs. It adds realizations of measurement noise to each of these sets of model outputs. Random numbers that represent noise have means of zero. For each observation, the standard deviation of measurement noise is calculated as the inverse of the weight assigned to that observation in the partial PEST control file that PDC1 reads. Alternatively, if a standard deviation is supplied in the fifth column of this file, this standard deviation is employed directly.

Observation transformations

If PDC1 is asked to read an observation transformation file, then all observations and their model generated counterparts are first multiplied by observation-specific scale values. The offset is then added. Optionally, log transformation follows that. Note however, that random realizations of measurement noise are added to model-generated counterparts to observations before they undergo transformation. In some circumstances, it is possible that this may render the value of a scaled and offset observation negative. This makes log-transformation impossible. If log transformation is requested under these circumstances, PDC1 will notify you of this and cease execution. Similarly, if a scaled and offset observation is negative prior to requested log transformation, then PDC1 will also object.

Observation subsets

As discussed above, PDC1 ignores observations whose weights are zero or negative. Alfonzo and Oliver suggest that when conducting analysis of prior data conflict, it is often a good idea to do this sequentially using different subsets of the entire observation dataset, particularly if that dataset contains observations of different types. They point out that a modeler is often able to make better use of the information that PDC analysis yields, if he/she can identify those particular aspects of an observation dataset that are in conflict with assumptions pertaining to model structure and parameters. Selective PDC analysis is easily undertaken using PDC1 by setting certain observation weights to zero in the partial PEST control file that is read by PDC1, and/or by selecting different observation groups for analysis in response to its prompts.

19 DSI by Claude

This chapter documents three programs that were written by Claude. Claude also wrote a manual for these programs; its manual is reproduced in this section. I have altered what Claude wrote only slightly. For ease of recognition, some of my alterations and notes are in italics and labelled as having been written by me.

I, and others, have used these programs and have been impressed by them.

As you will find when you read this chapter, Claude discusses a difficult case that was used for testing purposes. This case contains over 10,000 observations and a small number of predictions. Problems were encountered because of the size of the matrices that were required for data assimilation, as well as the fact that the training dataset contained observations of different types. The best way to solve this particular problem (and others like it) is to use DSI2 and POSTDSIMOD (documented above) instead of the programs that are documented in this section.

However, not all cases are as difficult as that discussed in the example. Functionality provided by the Claude-written programs that are documented in this chapter surpasses that provided by DSI2 in some ways, and far surpasses that provided by DSI1.

Note also that the ability to separate noise standard deviations from weights can be important in some data assimilation contexts (see below). This is supported by DSI2, but not by programs documented in this chapter.

19.1 Introduction

The three programs that are documented in this section implement data space inversion (DSI). They take an ensemble of model outputs that has already been generated — one realization per row of a CSV file — and use it to condition predictions on a set of measurements, without ever running the model again.

That is the central idea and it is worth being explicit about. Conventional inversion adjusts model parameters until the model reproduces the measurements. DSI never touches the parameters. It works entirely in the space of model outputs: the training ensemble has already absorbed whatever nonlinearity the model contains, and all that remains is a statistical relationship between the quantities you have measured and the quantities you want to predict.

What you gain is speed. Once the setup file is built, conditioning on a new dataset is a single matrix-vector product, so repeated predictions against different measurement sets cost almost nothing. What you pay is an assumption: the relationship is treated as linear and the ensemble as jointly multi-Gaussian. Much of this section is about understanding when that assumption is doing you harm, and what to do about it.

19.1.1 The Three Programs

Program	Purpose
`DSI_SETUP`	Reads the training ensemble and a PEST-style "partial PEST control file"; this is a file that contains, at a minimum, the "observation data" section of a PEST control file. Computes everything that does not depend on the measured values — the gain matrix and a conditioned ensemble — and stores it in a binary setup file. Writes a report describing what it found.
`DSI_PREDICT`	Reads a setup file written by DSI_SETUP and a set of measurements, and writes the posterior distribution of the predicted variables. One matrix-vector product.
`DSI_XVAL`	Leave-one-out cross validation of a setup file. Removes each training realization in turn, rebuilds the statistics from the remainder, predicts the held-out realization and compares against its known truth. This is the only honest measure of whether the method is working.

The input dataset for programs that are documented herein is the same as that required by programs DSI1, DSI2 and DSI3 of the PEST suite.

Read section 19.8.2 before you trust any output. The single most common way to be misled by these programs is to validate them on a dataset that is itself one of the training realizations. The result looks superb and means nothing.

19.1.2 The Normal Workflow

The following block shows program names and optional command line arguments. (Note that if you do not provide command line arguments, then the programs prompt you for their input data requirements.)

```
dsi_setup   training.csv  control.pst  setup.bin   [report.rpt]
dsi_xval    setup.bin     summary.csv  [detail.csv]
dsi_predict setup.bin     newdata.pst  results.csv [ensemble.csv]
```

Run them in that order. The cross validation comes before the prediction, not after it, because its job is to tell you whether the prediction is worth having, and to help you choose the options described in sections 19.5 and 19.6.

19.1.3 Command Line and Prompts

File names may be given on the command line in the order shown above. Anything that you leave off is prompted for. The options — normal score transforms, subspace truncation — are always prompted, never positional.

Because the prompts are read from standard input, the programs drive perfectly well from a redirected response file, which is how you would use them inside a batch process:

```
dsi_setup train.csv control.pst setup.bin rep.rpt < responses.txt
```

One caution: if a response file runs out of answers the program stops with a Fortran runtime error rather than a helpful message. Count your lines.

19.2 Input Files

19.2.1 The Training Ensemble (CSV)

One row per realization, one column per variable, with a header row of variable names. A leading column named *realization* is recognized and discarded. Columns that do not correspond to names that are provided in the control file are ignored, and the program reports how many it ignored.

*Comment by John. The protocol for the CSV file is therefore the same as that required by PEST-suite DSI programs. However, Claude-written programs that are documented herein do not require that observations/predictions be provided in the same order in columns of the CSV file as the order in which they appear in the partial PEST control file. Matching is by name, and not position. Note also that (in contrast to PEST-suite DSI programs), Claude-written programs that are documented in the present section do not optionally allow realizations to reside in **columns** of a CSV file; they must occupy **rows**.*

There is no limit on the number of columns or on the width of a line in the CSV file. The reader measures the file before reading it and sizes its buffers to what it finds, so a file with tens of thousands of columns, and with records hundreds of kilobytes wide, is read correctly.

19.2.2 The Partial PEST Control File

The first non-blank line is ignored — it is normally the section header. Each subsequent line carries

```
NAME   VALUE   WEIGHT   [OBSERVATION_GROUP]
```

and reading stops at the end of the file or at the next line beginning with an asterisk. Weights are interpreted as PEST does, so sigma = 1/weight.

Comment by John. Programs that are documented in the present section do not tolerate a full PEST control file. In contrast, PEST-suite DSI programs will read the "observation data" section of any file; they simply ignore information that does not reside in this section. Note also that Claude-written programs will not read data from an optional fifth column that contains measurement noise standard deviations.

The weight pattern defines the problem:

- Non-zero weight — a conditioning variable. These are the measurements.
- Zero weight — a variable to be predicted. Its *value* field is not used in the conditioning process, so it can carry a true value held back for testing, or anything at all.

Every named variable in the partial PEST control file must have a matching column in the CSV file, and no variable may appear twice. The number of conditioning variables may freely exceed the number of realizations; see section 19.6.

Variable names are held in 40 characters. A longer name is currently truncated without warning, which is worth knowing if your naming convention runs long.

19.3 What the Method Computes

19.3.1 General

You do not need to read this subsection to run the programs, but the output columns will not

mean much if you do not.

Write the training ensemble as two blocks: the conditioning variables and the predicted variables. Programs documented herein subtract the ensemble mean from each and divide by the square root of (number of realizations minus one), giving anomaly matrices $\mathbf{A}_o$ and $\mathbf{A}_p$. Let **R** be the diagonal matrix of measurement error variances implied by the weights. The prior covariances follow as products of anomaly blocks.

The linear-Gaussian (ensemble Kalman) update then gives the posterior mean and covariance of the predicted variables:

```
posterior mean        =  mu_p + G ( d - mu_o )
posterior covariance  =  Cpp - G Cpo'


where the gain          G = Cpo ( Coo + R )^-1
```

Two features of this arrangement explain nearly everything about how the programs behave.

19.3.2 The Gain Does Not Depend on the Measurements

G is built from the training ensemble and the weights alone. So is the posterior covariance matrix. Only the posterior mean involves the data, and it involves them linearly. This is why DSI_SETUP can precompute everything and DSI_PREDICT can be a single matrix-vector product.

It also has a consequence that surprises people: **the reported uncertainty is the same whatever the measurements say**. A dataset that fits the ensemble beautifully and one that fits it badly receive identical posterior standard deviations. Low quoted uncertainty is therefore never, by itself, evidence that a prediction is correct.

19.3.3 Conditioning Happens in Whichever Space is Cheaper

$\mathbf{C}_{oo}$ is as large as the number of conditioning variables squared, and its rank cannot exceed the number of realizations minus one. Where there are many more measurements than realizations, it is both enormous and singular. Programs that are documented herein therefore never form it. Using Woodbury's identity, with

```
M = I + Ao R^-1 Ao'        (number of realizations, squared)
W = M^-1 Ap


G(j,k)       = W(:,j) . Ao(:,k) / sigma_k^2
postsd(j)^2  = Ap(:,j) . W(:,j)
```

M is positive definite however many measurements there are, being the identity plus something positive semi-definite. The programs choose between this route and the direct one automatically, on whichever matrix is smaller, and report which they used. The answers agree to rounding; only the cost differs.

19.3.4 The Conditioned Ensemble

Rather than stopping at a mean and a variance, DSI_SETUP also stores a full ensemble of posterior realizations, generated by randomized maximum likelihood:

```
x_p(i,post)  =  x_p(i)  +  G ( d + eps_i - x_o(i) ) ,    eps_i ~ N(0,R)
```

Each member takes a prior realization, perturbs the measurements by a fresh noise draw, and applies the gain. The perturbation is not cosmetic realism: without it the ensemble comes out too narrow by exactly $\mathbf{GRG}^t$, so the noise is the term that restores the variance the gain removes. It must have covariance **R** for the cancellation to be exact.

Because the measurements enter as a shift common to every member, the ensemble is stored with that shift factored out and DSI_PREDICT simply adds it. The percentiles reported for your predictions are the percentiles of this ensemble, which means they are not forced to be symmetric about the mean and can carry genuine skewness from the training data.

19.4 DSI_SETUP

19.4.1 Prompts

The following prompts are issued by DSI_SETUP if it is run without any command line arguments.

Prompt	Meaning
Normal score transform the conditioning variables?	y / n — see section 19.5
Normal score transform the predicted variables?	y / n — see section 19.5
Truncate the ensemble subspace?	y / n — see section 19.6
Maximum dimensions retained (0 for no limit)	Only if truncating. An integer cap on retained directions.
Eigenvalue energy fraction retained (1 for all)	Only if truncating. Whichever of the two limits bites first applies.

A useful trick: answer *y* to truncation and then 0 and 1.0. Nothing is truncated, but the full eigenvalue spectrum is written to the report. That is the cheapest diagnostic these programs offer, and section 19.6 explains how to read it.

19.4.2 Outputs

- **The setup file** (binary) — everything DSI_PREDICT and DSI_XVAL need. Not human-readable. It carries a version stamp; a file written by an older version of the programs is refused with a message telling you to rebuild it, rather than being misread.
- **The report file** (text) — described below. Defaults to *dsi_setup.rpt* if you do not name it.

19.4.3 The Report File, Section by Section

19.4.3.1 Header block

Names of the input and output files, the counts of realizations, conditioning variables and predicted variables, the random seed, and the state of the three options. Check these first: the count of conditioning variables is the most common place to discover that a partial PEST control file does not say what you thought it said.

19.4.3.2 Note on the transforms

This appears only if a transform was applied. It warns that every mean and standard deviation in the report is expressed in the space that the conditioning happened in, which for a transformed variable is normal score space and not the units of the variable. DSI_PREDICT reports back-transformed values; the report does not.

19.4.3.3 Note on rank

This appears only when there are at least as many conditioning variables as realizations. It states the rank limit and explains that while the conditioning itself remains sound — the measurement noise makes ($\mathbf{C}_{oo}$ + $\mathbf{R}$) full rank — the information the ensemble can carry does not. Read it as a warning that the intervals below may be optimistically narrow, and that cross validation is now essential rather than optional.

19.4.3.4 Ensemble spectrum

This appears only if truncation was requested. One row per direction of the ensemble subspace:

Column	Meaning
`i`	Direction number, in descending order of eigenvalue.
`eigenvalue`	The eigenvalue of $\mathbf{A}_o\mathbf{R}^{-1}\mathbf{A}_o^t$ for this direction. These are the squared singular values of the noise-scaled anomaly matrix.
`sqrt(eig)`	Its square root — a signal-to-noise ratio for this direction. A value of 100 means the data claim to determine this combination of outputs to one part in a hundred.
`var frac`	$\lambda/(1+\lambda)$: the fraction of the prior variance the data remove along this direction. A direction with λ well below one is barely informed; one reading 1.000000 is claimed to be perfectly determined.
`cum energy`	Running fraction of the total eigenvalue sum.
`kept`	Marked where this direction survives the truncation you asked for.

Only the leading 200 directions are listed; a note says so if more exist.

19.4.3.5 Conditioning variables (i.e. variables with non-zero weights)

Column	Meaning
`name, group`	As given in the control file.
`prior mean`	Ensemble mean of this variable.
`prior sd`	Ensemble standard deviation.
`obs sigma`	Measurement error standard deviation, 1/|weight. If the conditioning variables were normal-score transformed this has been rescaled to Gaussian space — see section 19.5.3.
`sd/sigma`	The prior spread expressed in units of the measurement noise. This is the informativeness of the observation: large values mean it should strongly constrain the ensemble, values near or below one mean it carries little weight. It is the quickest sanity check you have on a weighting scheme.

19.4.3.6 Predicted variables (i.e. variables with zero weight)

Column	Meaning
`prior mean, prior sd`	Ensemble mean and standard deviation before conditioning.
`post sd (an)`	The analytic posterior standard deviation: the square root of the j-th diagonal of $\mathbf{A}'_p\mathbf{M}^{-1}\mathbf{A}_p$, equivalently of $\mathbf{C}_{pp} - \mathbf{C}_{po}(\mathbf{C}_{oo}+\mathbf{R})^{-1}\mathbf{C}^t_{po}$. Exact under the linear-Gaussian model.
`post sd (ens)`	The sample standard deviation of the stored conditioned ensemble for the same variable. A Monte Carlo estimate of the same quantity, computed by an entirely different route.
`var red %`	100 x (1 - post sd (an) / prior sd). How much of the prior uncertainty the conditioning claims to remove.

The line following the table reports the largest relative discrepancy between the analytic and ensemble columns. These two numbers are computed from separately coded quantities — the gain, the base ensemble and the analytic variance — so agreement is a consistency check on all three, and in particular a numerical check that the noise perturbation in the conditioned ensemble has exactly the right covariance. A few percent of difference between these quantities is sampling noise and harmless. A large discrepancy, or a systematic bias with the ensemble column always the smaller, would indicate a fault.

Treat *var red %* with suspicion. It is what the conditioning claims, not what it achieves. In the example described in section 19.9 it reads 98% for predictions that cross validation shows to be worse than useless.

19.4.3.7 Normality diagnostics

Skewness and excess kurtosis of every variable, computed on the raw training data before any transform, with a flag where the absolute skewness exceeds 0.5 or the absolute excess kurtosis exceeds 1. Where a transform was applied, two further columns give the same statistics afterwards; for a transformed variable they should be close to zero, which is a check

that the transform did what it claims.

The flags tell you whether a normal score transform is worth considering. They do not tell you whether it will help — see section 19.5.5.

19.5 Normal Score Transformation

19.5.1 What it Does

The linear-Gaussian update is exact only if the training ensemble is jointly multi-Gaussian. A normal score transform, or Gaussian anamorphosis, replaces each variable by the normal scores of its own empirical distribution: the values are sorted, assigned plotting positions $(i-0.5)/n$, and mapped to the corresponding quantiles of the standard normal. The conditioning then happens in that transformed space and the results are mapped back.

Repeated values are collapsed to a single table node carrying the mean of their scores, so the mapping is strictly monotone and invertible. Beyond the range of the training data the end intervals are extended linearly, so a measurement outside the training range is extrapolated rather than clipped.

19.5.2 What it Fixes and What it Does Not

The transform makes every marginal distribution Gaussian. It does not make the joint distribution Gaussian. A set of variables can have perfectly Gaussian marginals and still be related in a curved or tail-dependent way, and the covariance the gain is built from captures only the linear part of that relationship. This is the main limit on what the transform can buy you.

19.5.3 Why There are Two Separate Switches

The two sides are not alike, which is why they are controlled independently.

Predicted variables carry no measurement noise. Because the map is monotone, the posterior percentiles transform back exactly — the 5th percentile in normal score space is the 5th percentile in the units of the problem. The mean and standard deviation do not transform back, since a nonlinear map does not carry them across, so DSI_PREDICT computes those as sample statistics of the back-transformed conditioned ensemble instead. Transforming predictions is, in this sense, clean.

Conditioning variables do carry noise, and its standard deviation arrives in raw units as 1/weight. A nonlinear map has no single derivative, so that sigma cannot be carried across exactly. A derivative taken at the measured value would make the gain depend on the data, which would destroy the precomputation the whole design rests on. Section 19.5.4 describes what is done instead.

19.5.4 The One Approximation

When the conditioning variables are transformed, each observation's *sigma* is rescaled by the ratio of its transformed prior spread to its raw prior spread. This holds *prior_sd/sigma* fixed, so each observation is deemed exactly as informative after the transform as before, and the gain remains precomputable. The factor is stored in the setup file so that DSI_PREDICT can rescale changed weights the same way.

This is the only approximation the transform introduces, and the report says so when it

applies. In testing it has cost nothing measurable in calibration.

19.5.5 Does it help? Guidance

The honest answer is that you should measure it rather than assume it, and DSI_XVAL is how. Experience with one real problem, however, gives a useful prior:

- **Transforming the conditioning variables was clearly worth it.** On a 28-observation problem that comprised part of the testing suite of programs that are documented herein transformation improved the cross-validated RMSE reduction from 44.0% to 47.7% with coverage unchanged. The mechanism is worth understanding: the transform straightens monotone curvature in the relationship between measurements and predictions before the linear gain is applied, and that is a genuine improvement to the predicted mean.
- **Transforming the predicted variables bought little.** Coverage moved from 89.3% to 89.0% on a problem that was already well calibrated. It makes the reported intervals properly asymmetric, which is the right thing to quote for a skewed forecast, but it does not by itself improve accuracy.
- **Neither fixed a badly over-conditioned problem.** On the problem of section 19.9, coverage was 1.6% without any transform, 1.5% with predictions transformed and 1.6% with both. Non-Gaussianity was simply not the binding constraint there.

A note from John. The "coverage" that is referred to here is the percentage of posterior samples of a prediction that fall within the calculated posterior 90% confidence interval. Ideally, this should approach 90%. However, under conditions that are discussed in the example below, in some circumstances "ideal" conditions may be grossly violated.

A note on comparing runs. If the predicted variables are transformed, DSI_XVAL's RMSE is expressed in normal score units and cannot be compared with an untransformed run. Coverage is unaffected by a monotone map and so compares directly. To test whether a transform improves accuracy in the units of the problem, transform the conditioning variables only and leave the predictions alone.

Cost. The transform tables are two arrays of (realizations x variables) doubles. For predictions this is negligible. For a problem with many thousands of conditioning variables it is not: in the worked example discussed below, the setup file grew from 54 MB to 143 MB.

19.6 Subspace Truncation

19.6.1 The Problem it Addresses

The prior covariance estimated from an ensemble of *N* realizations has rank at most *N-1*. If you have more measurements than realizations, the update is being asked to exploit correlations estimated from *N* samples across a far larger number of variables. Correlations estimated from *N* samples are known to roughly one over the square root of *N*; with thousands of measurements the spurious components accumulate, and the update treats them as signal.

The symptom is distinctive and alarming: the reported uncertainty collapses, the setup report claims very large variance reductions, and the predictions are wrong. The update has, in effect, fitted noise.

19.6.2 What Truncation Does

In the realization-space formulation the posterior can be written in terms of the eigenvalues λ of $\mathbf{A}_o\mathbf{R}^{-1}\mathbf{A}^t_o$, where λ/(1+λ) is the fraction of prior variance the data remove along each direction. Truncation retains only the leading *p* directions and leaves the rest at the prior.

That last phrase matters. Truncation does not discard the trailing directions' uncertainty; it declines to claim any information about them. So *p* = 0 reproduces the prior exactly and *p* = full rank is the untruncated answer, with everything in between a controlled regularization.

19.6.3 The Two Controls, and Which to Use

- **Maximum dimensions retained** — a hard cap on *p*. Zero means no limit.
- **Eigenvalue energy fraction** — retain directions until the cumulative eigenvalue sum reaches this fraction of the total. 1.0 means no limit.
- Whichever bites first applies.

Use the dimension cap, and choose it by cross validation. The energy criterion is familiar from SVD-based inversion but it is close to useless here, and the spectrum shows why. In the worked example discussed below 90% of the eigenvalue energy sits in the first direction alone and 99.4% within the first fifty, so any energy threshold above about 0.9 still retains hundreds of directions whose *var frac* reads 1.000000 — directions the update believes are perfectly determined on the evidence of 520 realizations. Energy measures how large the eigenvalues are. The question you actually need answered is how far down the spectrum they remain trustworthy, and energy does not speak to that.

19.6.4 Choosing *p*

Give DSI_XVAL a list of candidate dimensions and it will evaluate all of them in a single pass. The eigen-decomposition does not depend on *p*, so the extra values cost almost nothing:

```
Dimensions to evaluate (blank for the setup value): 1,2,5,10,20,50,100,200
```

Then read the sweep table. The honest optimum is the largest RMSE reduction you can obtain while mean coverage stays near 90% and the standard deviation of the standardized error stays near 1.0. In a badly over-conditioned problem, you will see coverage fall monotonically as *p* rises, and the standardized error grow — in the following worked example from 1.01 at *p* = 1 to 87 at full rank.

Watch the worst columns, not only the means. A healthy mean reduction beside a worst value near or below zero means the gain belongs to a few variables while the rest learn nothing.

19.6.5 What Truncation Can and Cannot Fix

Truncation is a calibration fix. In the worked example discussed below, it moved coverage from 1.6% to 90.4% — the intervals became honest. It did not recover predictive skill: at no value of *p* did the full measurement set approach what a small subset of the same measurements achieved.

Comment from John. Claude is not making this very clear. What it is trying to say is that when (in the example below) only a few heads were used to condition the predictions, the conditioning process worked well by all criteria (good coverage statistics and significant reduction in posterior predictive uncertainty). However when these were combined with a

large number of AEM measurements (i.e. measurements of dbdt), every statistic deteriorated. This is a problem, because extra data should not induce worse predictive performance in a Bayesian world. See below.

The reason is that the cut is global. If one block of measurements dominates the spectrum, the leading directions are whatever that block constrains, and the directions carrying other measurements' information fall below any useful cut. There is a striking demonstration in section 19.9. If you find truncation fixing your calibration but not your accuracy, the problem is upstream, in the weighting.

19.6.6 Cost

Truncation requires an eigen-decomposition. In DSI_SETUP that is one decomposition and costs a second or two at 520 realizations. In DSI_XVAL it is one per held-out realization, which took about 6.5 minutes for a 520-realization problem and about 20 minutes for a larger one. Untruncated cross validation needs no decomposition and takes seconds, so if your weighting is such that you do not need truncation, you get the validation very much more cheaply.

19.7 DSI_PREDICT

19.7.1 Inputs and Checks

The "new observation file" that DSI_PREDICT requests is a partial PEST control file in the same format as the one used for setup. (There is no reason why these cannot be the same file.) Its non-zero-weight rows supply the measurements; its zero-weight rows are the predictions, and whatever values they carry are reported back as *file_value* for convenience.

The program checks that the zero / non-zero weight pattern matches the one the setup file was built for, and stops if it does not, rather than returning a wrong answer. If the weight values themselves have changed, DSI_PREDICT says so and rebuilds the gain and the conditioned ensemble, which is slower but correct.

DSI_PREDICT also reports the largest departure of any measurement from the prior mean, in units of the prior standard deviation, and lists any measurement beyond three. Those are the measurements your ensemble may not span, and the predictions are extrapolations with respect to them.

19.7.2 The Results File

Column	Meaning
`name, group`	The predicted variable, from the partial PEST control file.
`prior_mean`	Ensemble mean before conditioning, in the units of the variable.
`prior_sd`	Ensemble standard deviation before conditioning.
`post_mean`	Posterior mean after conditioning.
`post_sd`	Posterior standard deviation.
`post_p05`	5th percentile of the conditioned ensemble.
`post_p50`	50th percentile — the posterior median.
`post_p95`	95th percentile.
`file_value`	Whatever the zero-weight row of the observation file carried. Meaningful only if you put a true value there.
`std_error`	(file_value - post_mean) / post_sd. Meaningful only if *file_value* is a true value held back for testing. A value beyond about 2 says the prediction missed by more than the quoted uncertainty allows.

19.7.3 Where those Numbers Come From

This differs depending on whether the predicted variables were transformed, and it is worth knowing which you are looking at.

Column	No prediction transform	Prediction transform applied
`post_mean`	Analytic: mu_p + G(d - mu_o)	Sample mean of the back-transformed ensemble
`post_sd`	Analytic	Sample standard deviation of the back-transformed ensemble
`post_p05/50/95`	Percentiles of the conditioned ensemble	Percentiles of the back-transformed ensemble, exact under the monotone map
`prior_mean, prior_sd`	Analytic	From the raw training values held in the transform table

Without a transform the analytic mean and standard deviation are used because they are exact and free of Monte Carlo noise. The ensemble mean would differ from the analytic mean only by the sample mean of the noise draws, which is error with no compensating benefit.

With a transform they are sample statistics of necessity. The back-transform is nonlinear, so the mean of the back-transformed members is not the back-transform of the mean. In fact the analytic mean in normal score space back-transforms to the posterior median in raw space — which is exactly what *post_p50* is. The gap between *post_p50* and *post_mean* in a transformed run is the skewness of the posterior showing itself.

19.7.4 The Ensemble File

Optional. A header row of *realization* followed by the names of the predicted variables, then one row per realization giving the full conditioned ensemble — back-transformed if a prediction transform was applied.

This is the output to use for anything the summary statistics cannot express: joint probabilities across several predictions, risk calculations, or any quantity that depends on the shape of the posterior rather than its first two moments. The members inherit the multivariate structure of the real training realizations, which a draw from a fitted Gaussian would not.

19.8 DSI_XVAL and the Interpretation of Results

19.8.1 What it Does

Each training realization is removed in turn. The ensemble statistics and the gain are rebuilt from the remaining realizations; the held-out realization's conditioning variables, plus a fresh realization of measurement noise, are used as data; the resulting predictions are compared against that realization's known true values.

This measures accuracy, and more importantly it measures whether the reported uncertainty is honest. It is the only output of these programs that can tell you that.

19.8.2 The In-Sample Trap

This deserves its own section because it is the easiest way to be badly misled. **If you validate DSI_PREDICT on a measurement set that is itself one of the training realizations, the result will look superb however badly the method generalizes.**

A real example. In a problem with 11,428 measurements and 520 realizations, the observation file was actually training realization 320, with no noise added. The same prediction, done both ways:

	post_mean	post_sd	truth	error, in sd
realization in the training set	320.04	5.81	320.19	0.03
same realization held out	229.94	5.81	320.19	15.53

Out of sample the posterior mean reverted to essentially the prior mean of 231.5 — the 11,428 measurements taught it nothing about a realization it had not seen. And note that *post_sd* is 5.81 in both rows, because the posterior spread does not depend on the data. The

same confident-looking uncertainty accompanied an error of 0.03 standard deviations and one of 15.5.

The mechanism is that with very large eigenvalues in every direction, the update is effectively solving for the ensemble coefficients that reproduce the observed data vector. If that vector came from an ensemble member, the answer is that member, and the forecast is returned almost exactly. It is nearest-neighbour identification wearing the clothes of Bayesian conditioning.

So: to test with DSI_PREDICT, build the setup file from a training ensemble with the test realization removed. Or simply use DSI_XVAL, which does this for every realization at once.

19.8.3 The Summary File, Single Dimension

When truncation is off, or on with a single dimension, the summary file carries one row per predicted variable.

Column	Meaning
`name, group`	The predicted variable.
`prior_rmse`	Root mean square error of the prior mean as a predictor, over all folds.
`post_rmse`	Root mean square error of the posterior mean, over all folds.
`rmse_reduction_pct`	100 x (1 - post_rmse/prior_rmse). The skill of the conditioning for this variable. Negative means conditioning made the prediction worse than quoting the prior mean.
`bias`	Mean signed error. A systematic offset rather than scatter.
`mean_std_error`	Mean of (truth - posterior mean)/posterior sd. Should be near zero.
`sd_std_error`	Standard deviation of that standardized error. Should be near 1.0. Greater than 1 means the posterior standard deviation is too small — the prediction is more wrong than it admits. Less than 1 means it is conservative.
`coverage90_pct`	Percentage of folds in which the truth fell inside the symmetric Gaussian 90% interval, posterior mean plus or minus 1.645 posterior standard deviations. Should be near 90.
`coverage90_quantile_pct`	The same, but using the shape of the conditioned ensemble rather than a symmetric Gaussian interval. See 19.8.5.

19.8.4 The Summary File, Sweep Over Dimensions

If you supply more than one dimension, the summary file instead carries one row per dimension.

Column	Meaning
`dimensions_asked`	The value you requested.
`dimensions_used_min / _max`	What was actually retained, across folds. These can be lower than requested, because the rank available and the energy criterion also apply.
`mean_rmse_reduction_pct`	Average over predicted variables of the per-variable RMSE reduction. Each variable is measured against its own prior, so this does not depend on the units or spread of any one prediction.
`worst_rmse_reduction_pct`	The least of those per-variable reductions. Watch this one.
`mean_coverage90_pct, worst_`	Mean and worst coverage of the Gaussian interval.
`mean_coverage90_quantile_pct, worst_`	Mean and worst coverage of the sampled-shape interval.
`mean_sd_std_error`	Average standard deviation of the standardized error. Near 1.0 is calibrated.

The reduction columns are deliberately computed per variable and then averaged, rather than pooled across variables. Pooling squares of quantities in different units lets whichever prediction has the largest spread dictate the answer. In the worked example a pooled figure read +3.60% where the per-variable mean was +2.42% with a worst of -0.98% — the apparent gain belonged entirely to one flux whose prior standard deviation was 333 against about 2 for the heads. **Re-run with a single dimension to obtain the per-variable breakdown once you have chosen *p*.**

19.8.5 The Two Coverage Columns

The Gaussian column asks whether the truth fell within the posterior mean plus or minus 1.645 standard deviations. The quantile column asks whether it fell within an interval of the same nominal probability but taking its shape from the conditioned ensemble — which, being a sample rather than a formula, can be asymmetric.

Comparing the two tells you whether the non-Gaussian shape the ensemble carries is doing you any good. If the quantile column is materially better, the shape is worth having. If the two track each other, your error does not lie in the shape of the posterior.

In testing on a problem with pronounced skewness in the forecasts, the shape information was genuinely present — the standardized 5th and 95th percentile offsets came out at -1.62 and +1.53 against the Gaussian -1.645 — yet the quantile coverage was about 1.5 points worse. Both intervals were slightly too narrow, because neither represents the error in estimating the gain itself, and the sampled one was narrower still. The lesson is that sampling addresses the shape of the posterior, not its width, and width was the binding problem.

19.8.6 The Detail File

Optional, and one row per fold per predicted variable: *realization*, *name*, *truth*, *post_mean*, *post_sd*, *std_error*, preceded by a *dimensions* column when a sweep is running. Use it to look at the distribution of errors rather than their summary — for example to check whether the

large errors are concentrated in particular realizations, which would suggest the ensemble does not span them.

19.9 A Worked Example, and What it Teaches

The following is from a real problem and is included because every number in it is a trap that the output columns, read carelessly, would have let through.

Two measurement sets conditioned the same 520-realization ensemble and the same 28 forecasts. One used 28 period-1 groundwater heads. The other used 11,428 variables — those same 28 heads plus 11,400 *dbdt* values (obtained during an AEM survey) and a flux — so it was a strict superset of the first.

	28 measurements	**11,428 measurements**
Setup report, var red %	about 44%	about 98%
Cross-validated coverage	89.3%	1.6%
sd of standardized error	about 1.0	about 87
RMSE reduction, per variable	+43.1%	-54.1%

Adding 11,400 measurements to a set that already contained the informative ones made the predictions worse than quoting the prior mean, while the setup report's claimed variance reduction rose from 44% to 98%. A correctly specified Bayesian update cannot be harmed by more data, so the fault lay in the covariance estimate: 520 realizations cannot support correlations across 11,456 variables.

Truncation restored the calibration completely — coverage returned to 90.4% at p = 10 — but recovered almost none of the skill. (“*Skill” is the extent to which the posterior uncertainty is less than the prior uncertainty.*) The best any dimension achieved was +2.96% against the +43% that 28 of the same measurements delivered. The enigma was solved by comparing outputs of two truncated runs: one conditioned on all 11,428 variables and one on the 11,400 *dbdt* values alone gave numerically identical answers. The 28 heads contributed nothing, because they contributed nothing to the leading directions.

The spectrum had said as much in advance. Its leading eigenvalue was 1.5e9, a claim that the data determine the leading combination of outputs to one part in 38,800, which is not a credible statement about field measurements. The *dbdt* weights averaged 1460 against 20 for the heads, and there were 422 times as many of them, so the *dbdt* block dominated the spectrum by orders of magnitude and the heads were invisible to any global cut.

The lessons, in the order they are useful:

- Read the spectrum before you trust a variance reduction. If *var frac* reads 1.000000 for hundreds of directions, the weights are claiming more than the ensemble can support.
- Check *sd/sigma* per observation group in the setup report. Grossly unequal values across groups, multiplied by grossly unequal group sizes, mean one group will dominate regardless of how informative the others are.

- Cross validate before you believe anything, and validate on realizations that are not in the training set.
- If truncation fixes your calibration but not your accuracy, stop adjusting *p* and go back to the weights. *Comment from John: What Claude means here is that if, in a case such as this, truncation provides valid coverage statistics but uncertainty reduction is still too low, the good coverage statistics are an outcome of uncertainty that is probably higher than it needs to be. This was achieved through truncation – which denied the conditional process access to some important information.*
- Weighting is a statement about precision. Asserting more precision than your ensemble size can certify does not produce a confident answer; it produces a confident wrong answer.

19.10 Suggested Procedure

A workflow that puts the cheap diagnostics first.

- **Look at the spectrum.** Run DSI_SETUP answering y/0/1.0 to the truncation prompts. Nothing is truncated and the full eigenvalue table is written. Ask whether the implied precision is credible, and whether *var frac* stays at 1.000000 for implausibly many directions.
- **Check *sd/sigma* by group** in the conditioning table. If one group dominates both in per-observation weight and in count, rebalance before doing anything else. Scaling a group's weights down by the square root of its size is a reasonable starting point for making groups contribute comparably.
- **Cross validate with no truncation.** It takes seconds. If coverage is near 90 and the standardized error near 1, your weighting is defensible and you may not need truncation at all.
- **If coverage is poor, sweep *p*** in a single DSI_XVAL run and choose the largest reduction available while coverage holds near 90.
- **Consider transforming the conditioning variables.** Measure it, with predictions left untransformed so the RMSE stays comparable.
- **Re-run with the chosen settings and a single dimension** to obtain the per-variable breakdown, and read the worst columns as well as the means.
- **Only now run DSI_PREDICT** on your real measurements, and remember that its quoted uncertainty is a property of the setup file rather than of the answer.

19.11 Known Limitations

- Variable names longer than 40 characters are truncated silently.
- The gain and the prediction-observation cross covariance are stored at (predictions x measurements). This is harmless when the number of predictions is small and costly when it is not.

- The Cholesky factorization and the symmetric eigensolver are compact implementations rather than blocked library routines. They are comfortable at several hundred realizations and would be the bottleneck at several thousand. *Comment from John. If you encounter difficulties with execution speed, let me know. It should not be difficult to make a version that links to MKL libraries that can parallelize repetitive numerical tasks.*
- The setup report lists every variable, which for a problem with thousands of measurements produces a very large file.
- If a redirected response file is exhausted, the programs stop with a Fortran runtime error rather than a clear message.
- Nothing in the method captures non-monotone or heteroscedastic dependence, a non-Gaussian joint distribution with Gaussian marginals, or any dependence of the posterior shape on the measured values. The data enter once, as a shift; curvature in the measurements is invisible at any ensemble size.

19.12 Choosing a PEST-Suite Alternative

This section is written by John.

I was able to obtain satisfactory results with the difficult problem described above by using DSI2 from the PEST suite. As Claude points out, there are over 10000 dbdt measurements and just a few head measurements and a single historical flux measurement in the conditioning dataset. When everything was weighted according to measurement noise, dbdt measurements dominated the data assimilation process because there are so many of them.

When running PEST to calibrate the statistical model built by DSI2, I adjusted weights of the three observation groups for roughly equal visibility of each in the overall measurement objective function. However, as described in documentation of DSI2, I added an extra column to the partial PEST control file that is read by DSI2. This column contained measurement noise standard deviations. Weighting for visibility of each observation group ensured that the information content of no observation group dominated that of any other. Meanwhile, observation noise statistics were used for assessment of posterior uncertainty. So when POSTDSIMOD calculated posterior uncertainty, it got it right.

In contrast to the situation described by Claude above, the addition of lots of dbdt data did, in fact, reduce posterior predictive uncertainty well below that which was achieved using heads and flux alone. Furthermore, the estimated value of the prediction was closer to the true value of the prediction than the calculated value of its posterior standard deviation, despite the fact that posterior uncertainty is calculated to be relatively small.

19.13 A Note on Randomized Maximum Likelihood

The conditioned ensemble is a randomized maximum likelihood (RML) sample. Writing $\mathbf{x}$ for the joint vector of conditioning and predicted variables and $\mathbf{H}$ for the operator that selects the observable block, RML draws a prior sample $\mathbf{x}(i)$ and a perturbed dataset $\mathbf{d}(i) = \mathbf{d} + \mathbf{eps}_i$, then minimizes

```
J(x) = 1/2 (x - x(i))' C^-1 (x - x(i))  +  1/2 (Hx - d(i))' R^-1 (Hx - d(i))
```

Setting the gradient to zero gives

```
x(i,post) = x(i) + C H' ( H C H' + R )^-1 ( d(i) - H x(i) )
```

whose prediction block is exactly the update used here. Each posterior member is therefore the maximum-likelihood point of a randomized objective, which is where the name comes from.

In general, RML is an exact sampler only when the forward operator is linear; for a nonlinear operator the minimizers are posterior samples only up to a Jacobian-determinant weight. In data space that caveat disappears, because **H** is a selection matrix and perfectly linear by construction. The nonlinearity of the underlying model has already been absorbed into the training ensemble. This is precisely the bargain DSI offers: exactness in the sampler, in exchange for the multi-Gaussian assumption on the ensemble.

One consequence is worth drawing out. Because the update is affine in the samples, the mean of the conditioned ensemble equals the analytic mean plus the sample mean of the noise draws — exactly. The ensemble's non-Gaussian content therefore lives entirely in its higher moments; its mean is pinned to the linear-Gaussian answer however nonlinear the problem may be. Sampling gives you the shape of the posterior, not a nonlinear estimate of its centre.

20. References

Alfonzo, M. and Oliver, D.S., 2019. Evaluating prior predictions of production and seismic data. *Comput. Geosci.*, 23: 1331-1347.

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.

Doherty, J., 2015. Calibration and uncertainty analysis for complex environmental models. *Watermark Numerical Computing*, Brisbane, Australia. 227pp. ISBN: 978-0-9943786-0-6 Downloadable from www.pesthomepage.org.

Doherty, J. and Hunt, R.J., 2009. Two statistics for evaluating parameter identifiability and error reduction. *Journal of Hydrology*. 366, 119-127.

Hadi, A.S., 1992. A new measure of overall potential influence in linear regression. Computational Statistics and Data Analysis, 14, 1-27.

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.

Koch, K-R. 1999. Parameter Estimation and Hypothesis-Testing in Linear Models. Springer, Berlin, Heidelberg.

Lima, M.M., Emerick, A.A. and Ortiz, C.E.P., 2022. Data-space inversion with ensemble smoother. *Comput. Geosci.*, 24: 1179-1200.

Sun, W. and Durlofsky, L.J., 2017. A new data-space inversion procedure for efficient uncertainty quantification in subsurface flow problems. *Math. Geosci.* 49:679-715.

White, J.T., Doherty, J.E. and Hughes, J.D., 2014. Quantifying the predictive consequences of model error with linear subspace analysis. *Water Resour. Res, 50(2): 1152-1173.* DOI: 10.1002/2013WR014767

Yager, R.M., 1998. Detecting influential observations in nonlinear regression modelling. *Water Resourc. Res*., 34:7, 1623-1633.