

Secondary Identification and /or Statement of Cooperation

Approaches in Highly Parameterized Inversion: KeyPEST, A Keyword Input Reader for PEST++ and PEST

By Michael N. Fienen, Randall J. Hunt, and Jeremy T. White

Report Series XXXX–XXXX

U.S. Department of the Interior

U.S. Geological Survey

U.S. Department of the Interior

KEN SALAZAR, Secretary

U.S. Geological Survey

Marcia K. McNutt, Director

U.S. Geological Survey, Reston, Virginia: 201x  
Revised and reprinted: 201x

For more information on the USGS—the Federal source for science about the Earth,  
its natural and living resources, natural hazards, and the environment—visit   
<http://www.usgs.gov> or call 1–888–ASK–USGS

For an overview of USGS information products, including maps, imagery, and publications,  
visit <http://www.usgs.gov/pubprod>

To order this and other USGS information products, visit <http://store.usgs.gov>

Suggested citation:  
Author1, F.N., Author2, Firstname, 2001, Title of the publication: Place of publication   
(unless it is a corporate entity), Publisher, number or volume, page numbers; information   
on how to obtain if it’s not from the group above.

Any use of trade, product, or firm names is for descriptive purposes only and does not imply   
endorsement by the U.S. Government.

Although this report is in the public domain, permission must be secured from the individual   
copyright owners to reproduce any copyrighted material contained within this report.

Contents

Abstract 1

Introduction 2

Purpose and Scope 3

Design Concepts 4

Object-oriented Program (OOP) Design [taken from PESTCommander, needs revision] 4

KeyPEST Code Structure 6

Blocks 6

Keywords 6

Tables 7

Files 7

Using KeyPEST 8

Limitations of the Current Version of KeyPEST 8

Summary 8

References 9

Appendix 1: Minimal KeyPEST Keyword Input for PEST++ 12

PEST++ Additions to the PEST control file (taken from Welter and others, 2011). 12

Appendix 2: Additional KeyPEST Keyword Input for PEST 14

The PEST Control File [FROM WELTER AND OTHERS 2011/DOHERTY AND HUNT 2010 – NEED TO REVISE ALL OR SUBSET TO KEYWORDS] 14

Figures

1. Figure caption. 8

2. Figure caption with (*A*) multiple parts (*B*) that are divided by the (*C*) MultipartFigCap style. 8

Tables

1. This is a table title. 8

**Approaches in Highly Parameterized Inversion: keyPEST, A Keyword Input Reader for PEST++ and PEST**

By Michael N. Fienen, Randall J. Hunt, and Jeremy T. White

# Abstract

PEST (Doherty 2010a, 2010b) is a sophisticated parameter estimation software suite that has been developed over more than 20 years. As a result, the PEST input is relatively complex and includes many input values that are not commonly salient during the majority of today’s parameter estimation activities. A derivative of PEST – PEST++ (Welter and others, 2012) – was developed for simpler and more robust user access to the most-used capabilities of PEST. However, PEST++ currently uses the same complex input of PEST. This, in turn, has made for an unnessarily high learning curve for new PEST and PEST++ users. These overarching concerns are addressed with the code keyPEST, a keyword input reader and translator for PEST and PEST++. keyPEST is an object-oriented Python code that converts descriptive keyword input provided by a user into the appropropriate native (complex) format used by PEST and PEST++. The required number of keywords needed to initiate a PEST++ run is reduced by using a default-value-unless-overridden concept. Reasonable keyPEST default values are used to populate PEST and PEST++ input variables unless the user choses to override the default by including the appropriate keywords with new user-specified values. The code described here allows a reader to access the the complete set of PEST and PEST++ using descriptive keywords, but frees the user from needing to populate all required PEST and PEST++ variables. The intent is to provide a software platform that allows new users access to the powerful aspects of PEST and PEST++ with minimal initial training, while still allowing access to all PEST and PEST++ functionality as their parameter estimation needs increase. The code design itself is also intended to be extensible, to facilitate aumgentation as PEST and PEST++ codes are augmented and revised in the future.

# Introduction

PEST (Doherty 2010a, 2010b) is a sophisticated parameter estimation software suite that has been developed over more than 20 years. During the time of development, new insights and approaches have become standardized, both in computer programming and parameter estimation techniques. Most notably, many parameter estimation concepts have been extended, refined, and replaced, and more experience has been gained so that a subset of suggested approaches have emerged from the family of all possible methods (for example, Doherty, Fienen, and Hunt, 2010; Doherty and Hunt, 2010; Doherty, Hunt, and Tonkin, 2010). Given the, large established PEST user base, there is need to maintain backward compatibility, however. Backward compatibility has resulted in the legacy of the 20 years of PEST development still being reflected in the input format and and the amount of required input in the most current version. This legacy sustains a relatively complex input format ill-suited for user annotation. This, in turn, has made for an unnessarily high learning curve for new PEST users. To leverage the extensive experience gained during the development of PEST, the most-used capabilities of PEST, along with some enhancments too cumbersome to include in the original PEST, were re-coded in C++ (PEST++, Welter and others, 2011). PEST++ focused on tools well-suited for the majority of parameter estimation problems; by focusing on the most-used subset of PEST tools, it required much less input from the user—with an intent to keep PEST++ input to a minimum needed to access the powerful algorithms and approaches of PEST. However, in order to better serve the established large PEST user base , and to remain compatible with the host of utility programs that operate in conjunction with PEST, PEST++ inputs are entered as *additional* entries to the standard PEST control file (*\*.pst*); PEST++ functionality is effected by appending additional PEST++ specific keyword input into the existing an PEST control file (\*.pst) format, notifying the code of its presence with a “++” prepended to the beginning of each line in the input file containing PEST++ variables (Welter and others, 2011). PEST (version 12.3), in turn, has been modified to ignore PEST++ specific inputs. Therefore, the same PEST control file (\*.pst) file can be used by both PEST and PEST++. The result of this back-and-forth compatibility is that PEST++ currently requires all the complex minimum input of PEST, as well as additional input specific for PEST++, even though only a reduced subset of salient input is actually used by PEST++.

Although faciliting seamless access to PEST and PEST++, the input required of the user is still extensive, thus is considered cumbersome for reducing the learning curve for new PEST++ or PEST users. Therefore, there is a need for a method to align the minimum amount of user-supplied input for PEST++, while retaining options to provide all input required to access the full power of PEST. Moreover, the concept of reasonable-default-value-unless-overridden-by-the-user would help reduce the input required of new users, and simplify changes from defaults for expert PEST users. These overarching design concepts are addressed here with the code KeyPEST, a keyword input reader for PEST and PEST++. keyPEST is designed as an external utility that reads a simplified input file (discussed below), combines this input with default values for parameters not specified, and translates it into a format suitable for input into PEST and PEST++. The overall goal is to both make a concise and easy-to-read input file of minimum data needed from the modeler to perform parameter estimation, while retaining an as-needed access to all the power provided by the full suite of options provided by PEST and PEST.

# Purpose and Scope

This report focuses on providing software to facilitate implementation of parameter estimation guidelines provided by Doherty and Hunt (2010). These guidelines are founded on the use of a large number of parameters with soft-knowledge (Tikhonov) and subspace (singular value decomposition) methods for regularization in a hybrid approach that insures “the twin ideals of parsimony – simple as possible, but not simpler – are fully met.” The reader is directed to Hunt and others (2007), Doherty and Hunt (2009), Doherty and Hunt (2010), and Doherty, Hunt, and Tonkin (2010) for detailed discussion of these concepts. Similar to the stated goals of PEST++, the goals of keyPEST are: 1) to lower the barriers of entry for new users of parameter estimation software, 2) to develop efficient parameter estimation tools and algorithms appropriate for implementing the techniques discussed by Doherty and Hunt (2010) for solving highly parameterized problems; and 3) to provide an extensible software framework to support future development. This report documents the object-oriented design techniques to achieve these goals, a design approach not available in the coding of the original PEST. The programming language Python was chosen to provide coding efficiency and faciliate integration with the existing Python-based graphical user interface PESTCommander (Karanovic and others, 2012). This program has been name keyPEST , which reflects it being a keyword-oriented translator to obtain the required input for both PEST++ and the popular PEST parameter estimation code. Given the change in programming language and the inclusion of object-orientated design, the report includes material to facilitate the code being extended by more advanced users, by providing programming concepts in the main report body and included appendixes.

keyPEST does not encumber the user by requiring inputs for all PEST variables. Rather, the approach taken is to require of the user only the input needed to start the parameter estimation process while fully accessing the most widely used PEST and PEST++ capabilities. In order to reduce the amount of input, keyPEST relies on the concept of “reasonable default values”, which are values provided by the software developers intended to be suitable for the majority of parameter estimation problems. These values are default in that they are automatically used unless explicitly overridden by the user. The end result is requiring aminimum amount of input, thus making them easier to use. On the other hand, the ability to override default inputs allows the user access to all the options and functionality of PEST and PEST++. Therefore, the focus of this presentation is on getting the minimal input formulated and translated to PEST/PEST++. One typical example of overriding default parameters is shown to illustrate the concept; keywords needed to set all PEST and PEST++ parameters are provided in the Appendix.

Given this purpose and scope, the report is constructed for three objectives. Our primary purpose is to provide users, especially those new to PEST, with a simple description of the minimum keyword input for PEST++ users (Appendix 1). For use as a user becomes more experienced, the ability to override default, less-used, PEST input is also described using a similar keyward approach (Appendix 2). The focus of this presentation is on keyword-driven input; the original non-keyword PEST input is not included here for brevity, but can be found in Doherty (2010a, 2010b) and in the Appendix 1 of Doherty and Hunt (2010). The final objective is to convey the program design concepts and structure in order to facilitate extension of the code by others. This objective is the subject of the report body.

# Code Design

KeyPEST is developed in the object-oriented language Python. As with other object-oriented language, the objects provided in Python provide an efficient and extensible means of organizing and designing program elements. Python provides an extensive suite of modules, classes, exception handling, and other structures that can be readily used as building blocks for sophisticated programs, although new built-in modules are easily developed and integrated using Python and other programming languages.

keyPEST was developed using classes to both store the data used by the program and to organize methods that perform the actions specified by the progeram. In this way, the main program file keyPEST.py is kept brief and relatively simple. The broadest actions of the program are outlined in this file with details contained in the module keyPESTdata.py.

# keyPEST also takes advantage of the exception handling built in to Python. In this way, traceback errors that are displayed by Python are replaced by error messages defined in keyPEST. This makes interpreting the errors easier for users in that the nature of the errors is tied to the input supplied by the user. While not every possible contingency can be covered by such an approach, many possible mistakes and inconsistencies are handled and listed for the user.KeyPEST Input Structure

PEST and PEST++ input is translated by keyPEST from a user-supplied keyword ASCII text file, for presentation consistency given the file extension<casename>.*kyp*. The general structure of *\*.kyp* input file consists of sections of input, where the basic element of input entry is the ” input block”. The input block structure is designed based on a subset of the JUPITER protocol (Banta et al. **(2006)**). keyPEST is considered loosely based on the design of Jupiter; the full JUPITER protocolwas not invoked in keyPEST due to memory and computational overhead that can be problematic for large and complicated data sets – those typically used in highly parameterized inversion. XML input is also supported in keyPEST (Appendix 3); although the examples here focus on the Jupiter-like input design that is most easily read by humans, all PEST and PEST++ inputs can be accessed through the XML input scheme detailed in Appendix 3.

The strategy for keyPEST input is designed to use BLOCKS of ASCII text that are made up of either KEYWORDS for individual variables or TABLES for a series of data. The specification of whether a given block uses KEYWORDS or TABLES is predefined within keyPEST based on the type of data structure typically used to enter the data; the input BLOCKS defined below indicate which is required. TABLES, which often consist of long lists of observations or model parameters, can exist as ASCII text within the keyPEST *\*.kpy* input file, or can be called from external Excel *(\*.xls; \*.xlsx*) or ASCII text files. Further details are discussed below.

### Blocks [Need to make examples PEST++ rather bgaPEST in next 4 subheadings]

Input Blocks are allowed to take one of three?? forms: either KEYWORDS or TABLES. All input blocks are delineated by the words BEGIN and END. The header line also includes the name and type of the block and the final line contains the name of the block. For example:

BEGIN regularisation KEYWORDS

#PHIMLIM = 140

#PHIMACCEPT=150

FRACPHIM=1.0

MEMSAVE=memsave

WFINIT=3.5

WFMIN=42.0

WFMAX=44.5

LINREG=linreg

REGCONTINUE=continue

WFFAC =5.5

WFTOL=42.5

IREGADJ=4

NOPTREGADJ=2

REGWEIGHTRAT=1.2

REGSINGTHRESH=2.1

END regularisation

### Substantial integrity checking takes place to ensure that blocks are properly defined. For example, blocks may not in any way overlap. Also, the BEGIN and END lines must refer to the same block. Any text in the keyPEST input file that is not contained within a properly defined block is ignored.

### Comments are allowed (as is “commenting out” text) on the lines that define BEGIN and END of blocks, and within KEYWORDS blocks as discussed below. Comments are identified by placing a “#” symbol at the beginning of a comment line. For example, if a user is experimenting with the use of singular value decomposition (SVD), the entire SVD input block can be disabled by simply commenting out the beginning and end lines. As a result, all the text between the lines is considered by keyPEST to be outside of a valid input block and is therefore ignored.

### #BEGIN singular\_value\_decomposition KEYWORDS

### SVDMODE=1

### MAXSING=45

### EIGTHRESH=5.0E-07

### EIGWRITE+0

### #END singular\_value\_decomposition

To reactivate SVD, the leading “#” symbols can be removed and the entire block will be read.

### Keywords

Keyword variables correspond to single values identifies with an “=” sign. Multiple KEYWORDS can be entered on each line in an input file but no spaces are allowed in KEYWORDS names or variable values. An example is: PHIMLIM=140.0. Also note that comments can be added using the “#” symbol as the first character on a line. IN KEYWORDS blocks, any line beginning with “#” is ignored. This allows for annotations by the user, and for “commenting out” certain values in experimentation.

### Tables

Table variables are used for tabular data series that have multiple values in categories. Tables are identified by listing the number of rows (**nrow**), number of columns (**ncol**), and providing the keyword **columnlabels**. This is followed by **nrow** rows of data, with values arranged in **ncol** columns, corresponding to the same order as the **columnlabels**, and delimited by one or more spaces. For example:

BEGIN observation\_data TABLE

nrow=2 ncol=4 columnlabels

OBSNME OBSVAL WEIGHT OBGNME

ob1 50.0 1.1e-03 obsgroup1

ob2 42.0 1.1e-03 obsgroup1

END observation\_data

### Files

A user may want to shorten the length of the main input file by redirecting certain input to external text files. This can be done by signaling an input block with the word **FILES**, to read a file containing the entire set of information for the block. Only TABLE blocks may be redirected to external files, so the block type is always assumed to be a TABLE block. For example:

BEGIN parameter\_data FILE

parameters.txt

END parameter\_data

In this example, the contents of the file parameters.txt would be:

nrow=3 ncol=10 columnlabels

PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM

par1 log relative 3.09 1e-10 1e+10 group\_one 1.0 0.0 0

par2 tied relative 34.09 1e-10 1e+10 group\_one 1.0 0.0 0

par3 tied relative 4.09 1e-10 1e+10 group\_one 1.0 0.0 0

# 

# Using KeyPEST

[FIRST PARAGRAPH STANDALONE EXECUTABLE? OR PYTHON RUNTIME ENVIRONMENT? HOW DO THEY RUN IT?]

KeyPEST reads the keywords and translate the input into associated PEST/PEST++ control file (\*.pst) nonkeyword input. This requires the user to create a batch or script file that runs KeyPEST before calling PEST or PEST++, where the file input to PEST/PEST++ is the output file of KeyPEST. Alternatively, the user can specify the PEST++ socket whereby PEST++ is automatically called within the Python module. [Mike – see GMAN\_SOCKET for how Dave W. interfaced with GENIE]

# Limitations of the Current Version of KeyPEST

In order to retain programming power and flexibility of a higher level language, the runtime environment of Python makes the compilation of a standalone executable code more difficult. Thus, the user may need to have a working installation of Python in order to robustly access all features of KeyPEST. Other limitations of the current version of KeyPEST include:

* [MNF TO ADD]

# Summary

PEST (Doherty 2010a, 2010b) is a sophisticated parameter estimation software suite that has been developed over more than 20 years. As a result, the PEST input is relatively complex and includes many inputs that are not commonly varied during today’s parameter estimation activities. This, in turn, has made for an unnessarily high learning curve for new PEST users. PEST++ was developed by Welter and others (2011) to simplify required input for the most used capabilities of PEST, along with some enhancments too cumbersome to include in the original PEST. However, PEST++ is currently structured to use the same control file (\*.pst) as PEST. Thus, PEST++ currently requires all the complex minimum input of PEST, as well as additional input specific for PEST++.

These overarching concerns are addressed with the code KeyPEST, a keyword input reader for PEST and PEST++. KeyPEST is an object-oriented Python code that provides a means to convert keyword input into the appropropraite formate for the non-keyword input of PEST and, by extension, PEST++. This makes PEST++ more accessible to new users. The required number of keywords needed to initiate a PEST++ run is kept to a minimum by using a use-a-reasonable-default-value-unless-overridden-by-the-user concept. Default values can be overridden by including the optional keywords with new user-specified values. The code described here can translate the complete set of PEST++ functionality and a majority of PEST functionality. The code design is intended to be extensible, however, to facilitate aumgentation as PEST and PEST++ continue to be advanced in the future.

# References

Banta, E., E. Poeter, J. Doherty, and M. Hill (2006), JUPITER: Joint Universal Parameter

IdenTification and Evaluation of Reliability– An application programming

interface (API) for model analysis, Tech. Rep. U.S. Geological Survey Techniques

and Methods Book 6, Section E, Chapter 1, U.S. Geological Survey Techniques and

Methods Book 6, Section E, Chapter 1.

Doherty, J. 2010a. PEST, Model Independent Parameter Estimation. User Manual (5th ed., with slight additions). Brisbane Australia, Watermark Numerical Computing, 336 p.

Doherty, J. 2010b, PEST, Addendum to Model Independent Parameter Estimation. User Manual (5th ed., with slight additions). Brisbane Australia, Watermark Numerical Computing, variably paginated.

Doherty, J.E., Fienen, M.F., and Hunt, R.J., 2010,

Doherty, J.E., and Hunt, R.J., 2010,

Doherty, J.E., Hunt, R.J., and Tonkin, M.J., 2010,

Dongarra, J. (2011), Performance of Various Computers Using Standard Linear Equations Software. University of Tennessee, Knoxville, Tennessee, Computer Science Technical Report Number CS-89-85, url: [http:\\www.netlib.org\benchmark\performance.ps](http://www.netlib.org/benchmark/performance.ps)

Foster, I., 1995. Designing and Building Parallel Programs. Addison-Wesley Pearson Education, ISBN 9780201575941, 430 p

Harbaugh, A.W., 2005, MODFLOW-2005, the U.S. Geological Survey modular ground-water model -- the Ground-Water Flow Process: U.S. Geological Survey Techniques and Methods 6-A16, variously p.

Hunt, R.J., Luchette, J., Schreuder, W.A., Rumbaugh, J.O., Doherty, J., Tonkin, M.J., and Rumbaugh, D.B., 2010, Using a cloud to replenish parched groundwater modeling efforts: Rapid Communication for Ground Water, doi:10.1111\j.1745-6584.2010.00699.x.

MPI Forum. 2009. MPI: A Message-Passing Interface Standard. Version 2.2, September 4th

2009. available at: http:\\www.mpi-forum.org (Dec. 2009).

Schreuder, W.A. 2009. Running BeoPEST. In proceedings of the 1st PEST Conference, Potomac, Maryland, 1-3 November.

Schreuder, W.A., C. Muffels, M. Tonkin, J. Doherty, R.J. Hunt, and D. Welter. 2011. Efficient Use of Parallel Resources Using PEST. MODFLOW and More 2011, Integrated Hydrologic Modeling, International Ground Water Modeling Center, Colorado School of Mines, Golden, Colorado, June 6-8, 2011. Vol. 1. 787-791.

Zheng, Chunmiao, 2010, MT3DMS v5.3 Supplemental User's Guide, Technical Report to the U.S. Army Engineer Research and Development Center, Department of Geological Sciences, University of Alabama, 51 p.

Fienen N. Michael, Thomas C. Kunicki, and Daniel E. Kester, 2011, cloudPEST – A Python Module for Cloud-Computing, Deployment of PEST, a Program for Parameter Estimation, USGS, Open-File Report 2011–1062.

Hunt, R. J., Luchette, Joseph, Schreuder, W. A., Rumbaugh, J. O., Doherty, John, Tonkin, M. J., and Rumbaugh, D. B., 2010, Using a cloud to replenish parched groundwater modeling efforts. Ground Water, v. 48, no.3, p.360-365, doi: 10.1111\j.1745-6584.2010.00699.x

Muffels C., 2011, GENIE – Model Independent TCP\IP Run Management Software

Schreuder, W.A. 2009. Running BeoPEST. In proceedings of the 1st PEST Conference, Potomac, Maryland, 1-3 November.

Schreuder, W.A., C. Muffels, M. Tonkin, J. Doherty, R.J. Hunt, and D. Welter. 2011. Efficient Use of Parallel Resources Using PEST. MODFLOW and More 2011, Integrated Hydrologic Modeling, International Ground Water Modeling Center, Colorado School of Mines, Golden, Colorado, June 6-8, 2011. Vol. 1. 787-791.

# Appendix 1: Minimal KeyPEST Keyword Input for PEST++

The specific input blocks used in KeyPEST are discussed, in order of appearance in the <casename>.kyp file. For each block, data types are identified either as float, integer, or string. Values entered as float can include scientific/engineering notation, but in all cases should contain a “.” even if no fractional detail is included. Conversely, integers must not contain “.”. Variables identified as string may not include spaces because whitespace is used as the delimiter for rows in tables and separating keywords. Each block is also defined with a suffix of “cv” for “control variables,” or “data” for data

[DEW AND MNF TO FILL]

## 

## PEST++ Additions to the PEST control file (taken from Welter and others, 2011).

Information in the PEST control specific to PEST++ is specified on lines starting with “++”. Although the previous example places all the PEST++ input in a single section at the end of the PEST control file, this is not a requirement. This information does not need to be contiguous and can reside anywhere in the PEST control file. Lines starting with “++#” are considered comments and are ignored.

Unlike the rest of the PEST control file, PEST++ uses keywords rather than location to specify variables. Lines are parsed using the space, tab and parenthesis characters as separators. The example uses parenthesis to more clearly delineate the values assigned to the variable, but these could just as well be replaced by spaces. The following table includes a listing and explanation of the permissible PEST++ keywords.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Values** | **Description** |
| GMAN\_SOCKET | text | character string containing host and port separated by “:” | socket of the GENIE GMAN run manager. The socket contains the hostname and port of the GMAN run manager which will be used to make the model runs. For example if GMAN is running on the computer “my\_computer” listening to port 24772 then this variable should be specified as my\_computer:24772 |
| N\_ITER\_BASE | integer | 1 or greater | number of base parameter iterations performed for each super parameter iterations |
| N\_ITER\_SUPER | integer | 0 or greater | number of super parameter iteration performed for each base parameter iteration |
| SUPER\_EIGTHRES | real | any positive number (typically should be greater than 1.0e-7) | PEST++ will not include any super parameters whose ratio with the largest super parameter is less than this ratio. This value can as small as zero if the user wants to specify the number of super parameters solely with SUPER\_NMAX. As PEST++ uses SVD on the super parameter problem a low value for this SUPER\_EIGTHRES will not have an adversely impact the stability of the solution |
| SUPER\_NMAX | integer | integer between 1 and the minimum of maximum number of parameters and the maximum number of observations | maximum number of super parameters to use in the super parameter iterations |

# Appendix 2: Overriding keyPEST defaults - Additional KeyPEST Keyword Input for PEST and PEST++

## The PEST Control File [FROM WELTER AND OTHERS 2011/DOHERTY AND HUNT 2010 – NEED TO REVISE ALL OR SUBSET TO KEYWORDS]

For ease of reference, variables within the PEST control file are listed and the variables used by PEST++ are highlighted. PEST++ relies on the structure of the input file to deduce the parameters and only read the parameters that are absolutely necessary. For example, there is no need to read the NOBS variable as each line in the “observation data” section of the control file specifies an observation; however it is necessary to read the NPAR variable to know where specification of parameters ends and information on tied parameters begins. This is followed by short explanation of each variable used by PEST++.

pcf

\* control data

RSTFLE PESTMODE

NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM]

NTPLFLE NINSFLE PRECIS DPOINT [NUMCOM JACFILE MESSFILE]

RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE] [LAMFORGIVE]

RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND] [ABSPARMAX]

PHIREDSWH [NOPTSWITCH] [SPLITSWH] [DOAUI] [DOSENREUSE]

NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN] [PHIABANDON]

ICOV ICOR IEIG [IRES] [JCOSAVE] [VERBOSEREC] [JCOSAVEITN] [REISAVEITN] [PARSAVEITN]

\* automatic user intervention

MAXAUI AUISTARTOPT NOAUIPHIRAT AUIRESTITN

AUISENSRAT AUIHOLDMAXCHG AUINUMFREE

AUIPHIRATSUF AUIPHIRATACCEPT NAUINOACCEPT

\* singular value decomposition

SVDMODE

MAXSING EIGTHRESH

EIGWRITE

\* lsqr

LSQRMODE

LSQR\_ATOL LSQR\_BTOL LSQR\_CONLIM LSQR\_ITNLIM

LSQRWRITE

\* svd assist

BASEPESTFILE

BASEJACFILE

SVDA\_MULBPA SVDA\_SCALADJ SVDA\_EXTSUPER SVDA\_SUPDERCALC SVDA\_PAR\_EXCL

\* sensitivity reuse

SENRELTHRESH SENMAXREUSE

SENALLCALCINT SENPREDWEIGHT SENPIEXCLUDE

\* parameter groups

PARGPNME INCTYP DERINC DERINCLB FORCEN DERINCMUL DERMTHD [SPLITTHRESH SPLITRELDIFF SPLITACTION]

(*one such line for each of NPARGP parameter groups*)

\* parameter data

PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM

(*one such line for each of NPAR parameters*)

PARNME PARTIED

(*one such line for each tied parameter*)

\* observation groups

OBGNME [GTARG] [COVFLE]

(*one such line for each of NOBSGP observation group*)

\* observation data

OBSNME OBSVAL WEIGHT OBGNME

(*one such line for each of NOBS observations*)

\* derivatives command line

DERCOMLINE

EXTDERFLE

\* model command line

COMLINE

(*one such line for each of NUMCOM command lines*)

\* model input/output

TEMPFLE INFLE

(*one such line for each of NTPLFLE template files*)

INSFLE OUTFLE

(*one such line for each of NINSLFE instruction files*)

\* prior information

PILBL PIFAC \* PARNME + PIFAC \* log(PARNME) ... = PIVAL WEIGHT OBGNME

(*one such line for each of NPRIOR articles of prior information*)

\* predictive analysis

NPREDMAXMIN [PREDNOISE]

PD0 PD1 PD2

ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH

ABSPREDSWH RELPREDSWH

NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP

\* regularisation

PHIMLIM PHIMACCEPT [FRACPHIM] [MEMSAVE]

WFINIT WFMIN WFMAX [LINREG][REGCONTINUE]

WFFAC WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]

\* pareto

PARETO\_OBSGROUP

PARETO\_WTFAC\_START PARETO\_WTFAC\_FIN NUM\_WTFAC\_INC

NUM\_ITER\_START NUM\_ITER\_GEN NUM\_ITER\_FIN

ALT\_TERM

OBS\_TERM ABOVE\_OR\_BELOW OBS\_THRESH NUM\_ITER\_THRESH (*only if ALT\_TERM is non-zero*)

NOBS\_REPORT

OBS\_REPORT\_1 OBS\_REPORT\_2 OBS\_REPORT\_3..(*NOBS\_REPORT items*)

++# This line is a comment as are all lines that begin with “++#”

++# PEST++ input is parsed using key words that can be specified in any order

++ GMAN\_SOCKET(host:socket)

++ SUPER\_NMAX(max\_super) SUPER\_EIGTHRES(eig\_thres)

++ N\_ITER\_BASE(base\_iter) N\_ITER\_SUPER(super\_iter)

Variables in “control data” section of PEST control file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Values** | **Description** |
| RSTFLE | text | “restart” or “norestart” | instructs PEST whether to write restart data |
| PESTMODE | text | “estimation”, “prediction”, “regularisation”, “pareto” | PEST’s mode of operation |
| NPAR | integer | greater than zero | number of parameters |
| NUMCOM | integer | optional; greater than zero | number of command lines used to run model |
| RELPARMAX | real | greater than zero | parameter relative change limit |
| FACPARMAX | real | greater than one | parameter factor change limit |
| FACORIG | real | between zero and one | minimum fraction of original parameter value in evaluating relative change |
| PHIREDSWH | real | between zero and one | sets objective function change for introduction of central derivatives |
| NOPTMAX | integer | -2, -1, 0, or any number greater than zero | number of optimisation iterations |
| PHIREDSTP | real | greater than zero | relative objective function reduction triggering termination |
| NPHISTP | integer | greater than zero | number of successive iterations over which PHIREDSTP applies |
| NPHINORED | integer | greater than zero | number of iterations since last drop in objective function to trigger termination |
| RELPARSTP | real | greater than zero | maximum relative parameter change triggering termination |
| NRELPAR | integer | greater than zero | number of successive iterations over which RELPARSTP applies |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Values** | **Description** |
| MAXSING | integer | greater than zero | number of singular values at which truncation occurs |
| EIGTHRESH | real | zero or greater, but less than one | eigenvalue ratio threshold for truncation |
| EIGWRITE | integer | zero or one | determines content of SVD output file |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Values** | **Description** |
| PARGPNME | text | 12 characters or less | parameter group name |
| INCTYP | text | “relative”, “absolute”, “rel\_to\_max” | method by which parameter increments are calculated |
| DERINC | real | greater than zero | absolute or relative parameter increment |
| DERINCLB | real | zero or greater | absolute lower bound of relative parameter increment |
| FORCEN | text | “switch”, “always\_2”, “always\_3”, “switch\_5”, “always\_5” | determines whether central derivatives calculation is undertaken, and whether three points or four points are employed in central derivatives calculation |
| DERINCMUL | real | greater than zero | derivative increment multiplier when undertaking central derivatives calculation |
| DERMTHD | text | “parabolic”, “outside\_pts”, “best\_fit”, “minvar”, “maxprec” | method of central derivatives calculation |

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

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Values** | **Description** |
| OBGNME | text | 12 characters or less | observation group name |

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

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

Variables in “model command line” section of PEST control file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Values** | **Description** |
| COMLINE | text | system command | command to run model |

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

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

Variables in “prior information” section of PEST control file.

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

Variables in optional “regularisation” section of PEST control file.

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