Software Quality Assurance Plan for PEST

Tad S. Whiteside

June 2016

Q-SQP-G-00004, Revision 0

DISCLAIMER

Printed in the United States of America
Prepared for
U.S. Department of Energy

Keywords: Groundwater Modeling
PEST
Code Verification

Retention: *permanent*

Software Quality Assurance Plan for PEST

Tad S. Whiteside

June 2016

Prepared for the U.S. Department of Energy under contract number DE-AC09-08SR22470.

Savannah River National Laboratory™
OPERATED BY SAVANNAH RIVER NUCLEAR SOLUTIONS

Reviews and Approvals

G. P. Flach, Design Authority / Cognizant Technical Function — Date
Environmental Modeling

T. S. Whiteside, Design Check — Date
Environmental Modeling

C. G. Sherer, Cognizant Quality Function — Date
SRNL Quality Assurance

B. T. Butcher, Design Agency — Date
Environmental Modeling

D. A. Crowley, Design Agency Manager / Performing Manager — Date
Environmental Modeling

Contents

List of Abbreviations

CQF Cognizant Quality Function

CTF Cognizant Technical Function

DOE Department of Energy

EPA U.S. Environmental Protection Agency

GSA General Separations Area

HPC High Performance Computing

PA Performance Assessment

PEST Parameter ESTimation software

QA Quality Assurance

RMSD Root-Mean-Square Deviation

SQAP Software Quality Assurance Plan

SRNL Savannah River National Laboratory

SRNS Savannah River Nuclear Solutions

SWCD Software Classification Document

TCP/IP Transmission Control Protocol / Internet Protocol

SRS Savannah River Site

1 Introduction

The PEST "Parameter ESTimation" software is existing software and is classified as M&O/LW Level "C" software. As such, in compliance with Manual 1Q Procedure 20-1, this document, the Software Quality Assurance Plan (SQAP), is done as part of the required SQA Activities. The SQAP documents the Life Cycle Plans and the SQA Actions. The Life Cycle Plans document the Requirements, Design, Implementation, Testing, Installation and Acceptance, Operation and Maintenance, and Retirement of the software. Because PEST is existing software, these plans shall be created using a graded approach. The required SQA Actions are Configuration Control and Evaluation. The SQA Actions "Problem Reporting and Corrective Action" and "Risk and Safety Analysis" shall be done using a graded approach. The Cyber Security Analysis will be done using the requirements per Manuals 10Q and 7Q.

As described in the PEST manual (Doherty, 2016a), PEST was originally written to expedite the process of model calibration wherein values for model parameters are back-calculated by matching model outputs to measurements of system state. "Parameters" used within a model can represent the properties of the materials in which the processes simulated by the model take place, the stresses which initiate and support those processes, or both. What makes PEST different from the parameter estimation software which preceded it was the fact that PEST operates in a model-independent manner; it interacts with a model through the model's own input and output files. Hence no programming is required to use PEST to calibrate a model.

The model calibration is done by varying specified parameters and calculating the impact of those variations on the model output. PEST automatically continues the model variation until a specified number of iterations are complete or the convergence criteria are met. The result of this model calibration is a description of the uncertainty contained within the model parameters, which is an essential piece of information for models that support environmental decision-making.

2 Software Classification

Following Attachment 8.1 in Manual 1Q, Procedure 20-1, PEST is not exempt software and has been classified as Level "C" Software. This classification is recorded in Q-SWCD-A-00035, Rev 0. "Level C" software is generally associated with regulatory laws, environmental permits or regulations, and/or commitments to compliance. PEST will specifically be used in a supporting role with other software to provide reasonable assurance that the Performance Objectives of DOE Order 435.1 will be met.

3 SQA Procedures/Plans

To assure software quality, this document identifies the following information, and if required, where it can be found within the referenced sections:

- A description of the overall nature and purpose of the software - Section 5.1
- The software products to which it applies - PEST manual (Doherty, 2016a)
- The organizations responsible for performing the work and achieving software quality, including tasks and responsibilities - Not Applicable because this is externally developed, Existing Software
- The following:
  - software engineering methods - Not Applicable because this is externally developed, Existing Software
  - software life cycle phases and requirements for each phase - Section 5
- A risk and safety analysis - As appropriate within each Life Cycle subsection
- The required documentation to be maintained - As appropriate within each Life Cycle subsection
- The standards, conventions, techniques, or methodologies that shall be used to guide the software development, as well as methods to assure compliance to the same - Not Applicable because this is externally developed, Existing Software
- The required software review - Section 5.4
- The methods for error reporting and corrective action - Section 6.3
- Software configuration control requirements per this procedure and methodologies used - Section 6.1
- The security capabilities to be implemented, using approved cyber security procedures and guidance - As appropriate within each Life Cycle subsection

4 Roles and Responsibilities

Oversight and assignment of all PA related work, including PEST code use, is provided by the Performing Manager, Environmental Modeling. The Cognizant Technical Function (CTF) associated with PEST code use is Greg Flach of the Environmental Modeling group of SRNL. The Cognizant Quality Function (CQF) is the SRNL Quality Assurance group. Should other SRNL business programs choose to use PEST code at the same Level "C" functional classification, oversight will be

provided by the appropriate Performing Manager. The CTF, CQF, and this SQAP can remain unchanged.

5 Life Cycle Plans

Manual 1Q, Procedure 20-1 will be used to assure quality throughout the Software Life Cycle. Software specific implementation of 1Q, 20-1 for PEST is discussed below as needed to supplement 1Q, 20-1.

5.1 Requirements

PEST is existing software and was chosen for its significant user base and for the following capabilities to produce inverse solutions. These are the capabilities of interest for GSA flow modeling: parameter-estimation/calibration, prediction, and uncertainty analysis. PEST performs non-linear regression using a weighted least-squares method, and provides methods for dealing with parameter non-uniqueness such as Singular Value Decomposition and Tikhonov regularization. PEST interfaces with the flow simulation code through its existing input and output text file structures and formats. PEST offers the optimization capabilities needed to perform model calibration, making it a suitable choice.

5.1.1 Functionality

PEST is "Parameter ESTimation" software which uses a model control file to input parameter values to a model, run the model, parse the output, compare the results to previous runs, and continue running the model until the parameter values stabilize or the limit to the number of model runs is reached. The PEST manual (Doherty, 2016a) provides a full description of PEST's capabilities.

5.1.2 Performance

The performance of PEST is dependent on the number of parameters and number of model runs required to reach convergence. The main performance bottleneck is in the running of the model, which is independent of PEST. As existing software, we did not have any input on the design or implementation of algorithms used to perform it's functions, however based on inspection of the code and experience in trial-usage, performance is not an issue.

5.1.3 Design

Because PEST is existing software there are no constraints imposed on the design of the software.

5.1.4 Attributes

For PA purposes, PEST will run under the GNU/Linux operating system. PEST is written in FORTRAN and we have the source code and the capability to install all necessary external dependencies and compile the code to executable format.

5.1.5 External Interfaces

PEST is run by inputting a PEST control file. This PEST control file follows a specified format and must be constructed prior to running PEST. Once executing, PEST requires no further interactions with the user. PEST does not alter any hardware controls. PEST does interact with the model software, for example PORFLOW, by creating model input files, with varying input parameters, and then executing the model code. PEST can run on a single computer or on multiple computers and can communicate either through writing and reading of files or through TCP messaging.

5.1.6 Cyber-Security Analysis

In compliance with Manual 10Q, Procedure 300, PEST meets all cyber-security requirements. PEST runs under the end-user's account and can only read or write files in locations specified by the end user. In addition the TCP communication is only with user specified computers (IP addresses).

5.1.7 Risk and Safety Analysis

A more complete discussion the risk of using PEST (and models) alone to create a decision-support tool is found in the PEST manual (Doherty, 2016a). In summary, using PEST will not increase the risk of a poor environmental outcome and should provide greater confidence in the model results. Using PEST and expert analysis will not cause any additional safety concerns.

5.2 Design

Because PEST is existing software, the design portion of the software life cycle is not applicable to this SQAP. However, the source code for PEST is available to examine the design. Also, there are two user manuals (Doherty, 2016a,b) that provide a technical description of the software and how it interacts with input and produces output. In addition, these provide a description of how each module performs.

5.3 Implementation

Because PEST is existing software, the implementation phase of the software life cycle is not applicable. However, the source code for PEST is available to examine the implementation of the design. Also, should it be necessary to report a problem in the software, the code can be examined to see if there is a logic or other error that could help the author correct the problem.

5.4 Testing

Because PEST is existing software, it was tested by compiling and using the software. PEST was tested using three different problems in four different modes of operation. The computer code created to run the different test cases is included as Listing 10, in the Appendix.

PEST was compiled on Red Hat Enterprise Linux Workstation release 6.7 (Santiago). It was tested on a single computer (skink4) as well as multiple SRNL HPC compute-nodes within the "pople" queue.

PEST has several modes of operation and are described as follows. PEST has the capability of being run serially on a single processor-core ("s"), in parallel on a single machine using multiple processor-cores with a file-based communications system ("sp"), in parallel on a single machine using a TCP-based (Transmission Control Protocol) communications system ("sb"), in parallel on multiple machines using a TCP-based communications system ("hb"). These are the most probable use cases at SRNL for PEST and so were the ones tested to ensure the results produced are consistent. PEST also has the capability to be run in parallel on multiple machines using file-based communications, but the TCP based method is much faster and more robust, so this capability was not tested.

The TCP-based modes of operation ("sb" and "hb"), which use the BEOPEST version of PEST, produce identical results. The file-based modes of operation ("s" and "sp"), which use the PEST and PARALLELPEST versions of PEST, produce identical results. There are no differences between these two communication systems for case 5.4.3 (1 parameter to fit). With multiple parameters, cases 5.4.1 and 5.4.2, the differences between parameter estimates are on the order of 1×10^{-3}. This difference is within the tolerance set within the PEST control file.

A description of the steps necessary to run PEST is found within in Section 5.4.1, these steps will be abbreviated in Sections 5.4.2 and 5.4.3. Complete details of the various PEST-specific files and settings are found within the PEST manual (Doherty, 2016a).

Test case 5.4.1 was done to ensure PEST reproduces reported results. Test cases 5.4.2 and 5.4.3 test likely usage scenarios and how well PEST can converge on a solution for specified parameters.

5.4.1 Soil Specific Volume vs Water Content

This test case was taken from the PEST manual (Doherty, 2016a) to ensure PEST was compiled properly and the results found match those reported within the PEST manual.

In this experiment, the specific volume of a soil clod is measured at a number of different water contents as the clod is desiccated through oven heating. The results from this experiment are shown in Figure 5.4.1. From these observations, a model consisting of two line segments should provide a description of the soil physics. The line segment fitted through the points of low water content is referred to as the "residual shrinkage" segment; the segment covering the remainder of the data is referred to as the "normal shrinkage" segment, see Figure 5.4.2.

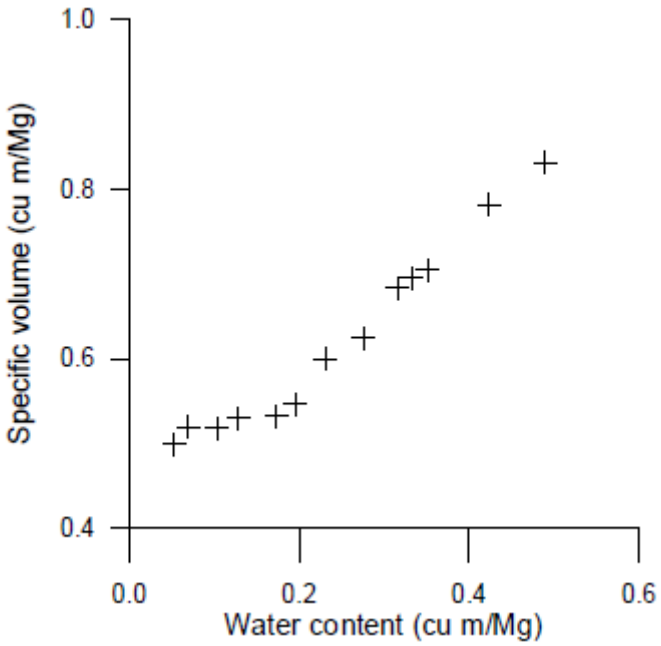

Figure 5.4.1: Soil clod shrinkage data.

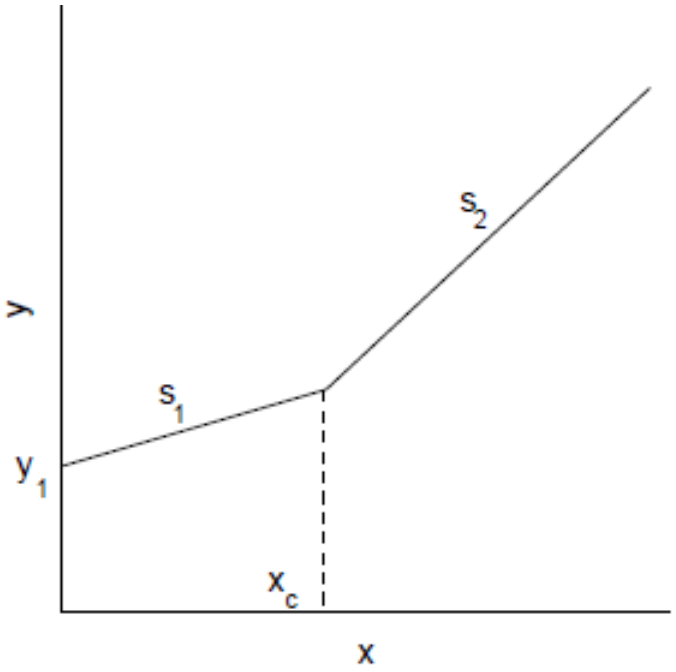

Figure 5.4.2: Soil clod shrinkage model.

The equation for the two-line system can be expressed as

$$y = s_1 x + y_1 \qquad x \leq x_c \qquad (5.4.1a)$$

$$y = s_2 x + (s_1 - s_2) x_c + y_1 \qquad x > x_c \qquad (5.4.1b)$$

where x is the water content, y is the specific volume, s_1 and y_1 are the slope and intercept of the low water content segment and s_2 is the slope of the higher water content segment and the x-coordinate of the point of intersection of the two line segments is x_c.

A simple FORTRAN computer program that implements this model reads an input file which supplies values for s_1, s_2, y_1, and x_c and the measured water contents values. This program writes an output file which lists both the input water content values and the calculated specific volumes, based on the parameter and input values. This program needs to be compiled with the command `gfortran -static -o twoline`, this ensures the necessary libraries are available within the executible (important when testing on multiple machines).

PEST adjusts the model parameters and automatically runs and reruns the model until the differences between the observed and calculated output values are minimized through least-squares analysis. The full mathematical details of how the number of model runs are chosen and the specific analyses done is found in the PEST manual (Doherty, 2016a).

To setup a model for use by PEST requires a series of steps. In order to simplify PEST's operation, the model may need to be wrapped in a script that calls the model and then a post-processing program to create a simple output file. The basic workflow to create a PEST-ready model is as follows:

1. Gather necessary files: input_file, model, output_file generated by model
2. Make the PEST instruction file. This file describes how to handle model output.
   (a) Copy the original output_file, Listing 1, to a new file called "output.ins"
   (b) Edit output.ins to tell PEST where to look for specific output, a complete description of how PEST parses the instruction file to look for specific output is found in the instruction manual in the section "Instruction Files" (Doherty, 2016a). Add the command "pif#" and then the appropriate instructions, see Listing 2.
   (c) Check if the instruction file is formed correctly by running the program "inscheck output.ins".
   (d) If you see any errors, fix those and rerun "inscheck output.ins"
   (e) If no errors run "inscheck output.ins output_file", this will create the file "output.obf" using the original model output file. Look at "output.obf" to make sure all the desired output observations are read correctly.
3. Copy "output.obf" to "measure.obf"
4. Edit "measure.obf" and replace the model output with measured output. In the two-line model this is the observed specific volumes.
5. Make the PEST template file. This file describes how to handle model input.
   (a) Copy the input_file, Listing 3, to a new file called input.tpl
   (b) Edit input.tpl, Add the command "ptf #" and replace parameter values with variables. This will tell PEST which parameter values to replace, see Listing 4.
   (c) Check the template file by running the program "tempchek input.tpl". This creates a file with all the parameter variables listed called "input.pmt".
6. Make the PEST parameter value file.
   (a) Copy "input.pmt" to "input.par"
   (b) Edit "input.par" so that it has the form like Listing 5. Where PRECIS is either "single" or "double", DPOINT is either "point" or "nopoint", p1 is the name of parameter 1, init_val is the initial value, SCALE is a multiplier of the initial value, and OFFSET is added to the initial value, such that the parameter value is of the form $p_1 = init\_val * SCALE + OFFSET$, see Listing 6.

(c) Check these files are correct by running "tempcheck input.tpl input.dat input.par", which creates the input.dat file based on the template and parameter files.

7. Make the PEST control file by running "pestgen twoline.pcf input.par measure.obf"

   (a) Edit the twoline.pcf by changing the following

      i. model → model.exe (eg twoline or model.sh or file that actual runs the model and postprocesses the output)

      ii. model.tpl model.inp → input.tpl input.dat

      iii. model.ins model.out → output.ins output.dat

      iv. other adjustments, such as parameter bounds can be done here as well, see the PEST user manual for full details.

   (b) Check the PEST control file, run "pestchek twoline.pcf"

   (c) If you see any errors, correct those.

   (d) If no errors, all is good, now PEST is ready to run, see Listing 7.

8. Run Pest, "pest twoline"

After running PEST, a run record file is created, which contains a record of the runs. In addition, a parameter file containing the estimated parameter set is also created. Figure 5.4.3 show the lines of best fit superimposed on the laboratory data.

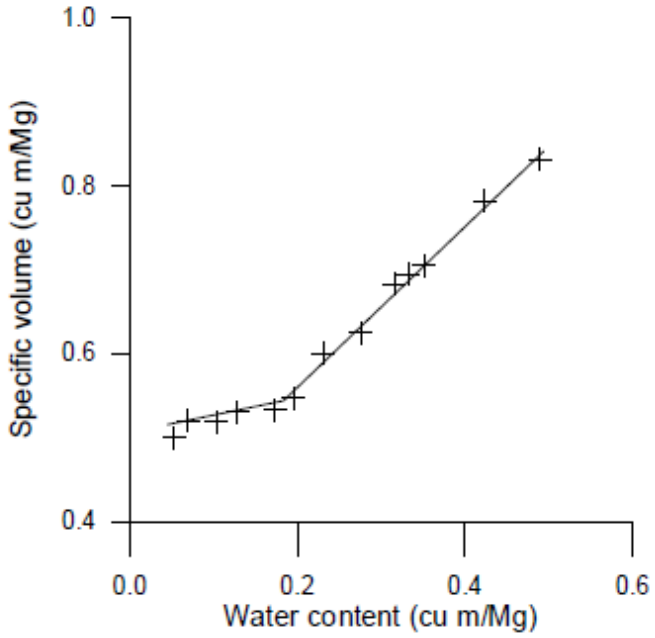

Figure 5.4.3: Soil clod shrinkage data with lines of best fit superimposed.

Table 5.4.1 shows the parameter values from the PEST manual, these can be compared with Table 5.4.2, which shows our results. There are minimal differences between the original and our results are are within 1×10^{-3}, which is how the PEST control file is setup.

As can be seen, the parameter values for the different modes of operation are also consistent to within 1×10^{-3}, which is how the PEST control file is set up. The ϕ value is within 1×10^{-5}. These differences can be accounted for by how PEST alters the parameter values while exploring the parameter space and are acceptable for the usage of PEST at SRNL.

Table 5.4.1: Parameter Values from PEST manual

Parameter	Value
ϕ	6.71E-4
s1	0.238
s2	0.963
y1	0.497
xc	0.174

Table 5.4.2: Showing the mode of PEST operation, the parameter name, and the parameter value. The ϕ parameter is the Sum of squared weighted residuals.

Mode	Parameter	Value
hb	phi	6.7098E-04
sb	phi	6.7098E-04
s	phi	6.7127E-04
sp	phi	6.7127E-04
hb	s1	0.2358606700000000
sb	s1	0.2358606700000000
s	s1	0.2393573700000000
sp	s1	0.2393573700000000
hb	s2	0.9625417100000000
sb	s2	0.9625417100000000
s	s2	0.9639391800000000
sp	s2	0.9639391800000000
hb	y1	0.4966959400000000
sb	y1	0.4966959400000000
s	y1	0.4964718000000000
sp	y1	0.4964718000000000
hb	xc	0.1733005600000000
sb	xc	0.1733005600000000
s	xc	0.1742852500000000
sp	xc	0.1742852500000000

Listing 1: output_file

```
0.520000E01  0.415600
0.680000E01  0.420400
0.103000     0.430900
0.128000     0.438400
0.172000     0.451600
0.195000     0.458500
0.230000     0.469000
0.275000     0.482500
0.315000     0.502000
0.332000     0.515600
0.350000     0.530000
0.423000     0.588400
0.488000     0.640400
```

Listing 2: output_ins

```
pif #
l1 (o1)19:26
l1 (o2)19:26
l1 (o3)19:26
l1 (o4)19:26
l1 (o5)19:26
l1 (o6)19:26
l1 (o7)19:26
l1 (o8)19:26
l1 (o9)19:26
l1 (o10)19:26
l1 (o11)19:26
l1 (o12)19:26
l1 (o13)19:26
```

Listing 3: input_file

```
0.3 0.8
0.4
0.3
13
0.052
0.068
0.103
0.128
0.172
0.195
0.230
0.275
0.315
0.332
0.350
0.423
0.488
```

Listing 4: The template file input.tpl

```
ptf #
# s1 # # s2 #
# y1 #
# xc #
13
0.052
```

```
0.068
0.103
0.128
0.172
0.195
0.230
0.275
0.315
0.332
0.350
0.423
0.488
```

Listing 5: The input parameter file input.par

```
PRECIS DPOINT
p1 init_val SCALE OFFSET
p2 init_val SCALE OFFSET
.
.
.
```

Listing 6: The actual input parameter file for two-line

```
single point
s1 0.3 1.0 0.0
s2 0.8 1.0 0.0
y1 0.4 1.0 0.0
xc 0.3 1.0 0.0
```

Listing 7: The actual PEST control file for two-line

```
pcf
* control data
restart estimation
4 13 4 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
s1 relative 0.01 0.0 switch 2.0 parabolic
s2 relative 0.01 0.0 switch 2.0 parabolic
```

```
y1 relative 0.01 0.0 switch 2.0 parabolic
xc relative 0.01 0.0 switch 2.0 parabolic
* parameter data
s1 none relative 0.300000 1.00000E+10 1.00000E+10 s1 1.0000 0.000 1
s2 none relative 0.800000 1.00000E+10 1.00000E+10 s2 1.0000 0.000 1
y1 none relative 0.400000 1.00000E+10 1.00000E+10 y1 1.0000 0.000 1
xc none relative 0.300000 1.00000E+10 1.00000E+10 xc 1.0000 0.000 1
* observation groups
obsgroup
* observation data
o1 0.501000 1.0 obsgroup
o2 0.521000 1.0 obsgroup
o3 0.520000 1.0 obsgroup
o4 0.531000 1.0 obsgroup
o5 0.534000 1.0 obsgroup
o6 0.548000 1.0 obsgroup
o7 0.601000 1.0 obsgroup
o8 0.626000 1.0 obsgroup
o9 0.684000 1.0 obsgroup
o10 0.696000 1.0 obsgroup
o11 0.706000 1.0 obsgroup
o12 0.783000 1.0 obsgroup
o13 0.832000 1.0 obsgroup
* model command line
model
* model input/output
model.tpl model.inp
model.ins model.out
* prior information
```

5.4.2 Multi-layer Hydroconductivity

The second test case for PEST uses Problem 4.3 from the PORFLOW QA (Whiteside, 2016), which describes a problem involving steady-state groundwater flow through a heterogeneous subsurface system, as seen in Figure 5.4.4.

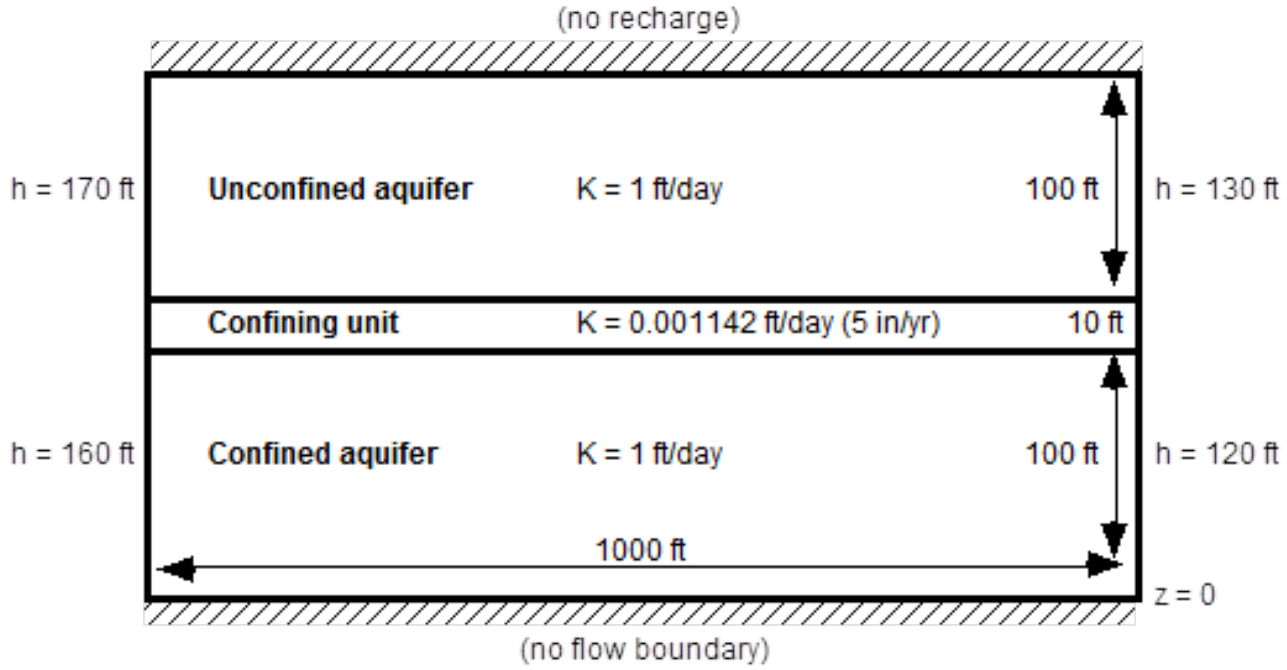

Figure 5.4.4: A Heterogeneous Subsurface System Consisting of an Unconfined Aquifer, Confining Unit, and Confined Aquifer.

There is a known analytical solution to this problem and the results of that solution are used as the observed values when creating the PEST model. The parameters being adjusted are the unconfined aquifer hydraulic conductivity, K_{xu}, the confined aquifer hydraulic conductivity, K_{xc}, and the confining unit's hydraulic conductivity, K_c. The analytical solution is based on a value of 1.0 for both K_{xu} and K_{xc} and a value of 1.142×10^{-3} for K_c. In this test problem, we set the K_{xu} value to 0.9, the K_{xc} value to 1.1, and the K_c value to 1.1×10^{-3}.

This model is more complicated than the other models and has a higher run time, so we limited testing to the "hb" mode of operation. Table 5.4.3 shows the results of PEST testing. The reported ϕ value, when divided by the number of observations (102) and square-rooted becomes the $RMSD = 9.9 \times 10^{-5}$, which is better than the equivalent RMSD (= 0.228) reported in (Whiteside, 2016). Where the equivalent RMSD is the combination of the $RMSD = 0.28$ for the confined aquifer and the $RMSD = 0.16$ for the unconfined aquifer.

Looking at the Sensitivity file, it should be noted the sensitivity of K_{xu} is 7.742×10^{-2}, K_{xc} is 1.60×10^{-1}, and K_c is 130.167. This shows that a small change in K_c can have a large impact on the overall performance of the model. An example of this impact can be seen in the following subtest.

As part of this test case, we also tested leaving K_c fixed at the initial analytical value of 1.142×10^{-3} and allowing K_{xc} and K_{xu} to vary, after setting those to 1.10 and 0.90 respectively. Under these conditions, these parameters converged to $K_{xu} = 1.216072$ and $K_{xc} = 1.151417$, with a ϕ value of

Table 5.4.3: Showing the mode of PEST operation, the parameter name, and the parameter value. The ϕ parameter is the Sum of squared weighted residuals.

Mode	Parameter	Value
hb	phi	1.0000E-06
hb	kxu	1.007823400000000
hb	kxc	1.007824100000000
hb	kc	1.1509399000000000E-03

0.7234. This ϕ value, when divided by the number of observations (102) and square-rooted becomes the $RMSD = 0.084$. This large difference shows the impact of the K_c parameter on the system.

5.4.3 Unconfined Aquifer Subject to Recharge and Drain Boundary Conditions

The third test case for PEST uses Problem 4.4.2 from the PORFLOW QA (Whiteside, 2016), which describes a problem where an unconfined aquifer experiences both recharge and drainage at the ground surface, as seen in Figure 5.4.5.

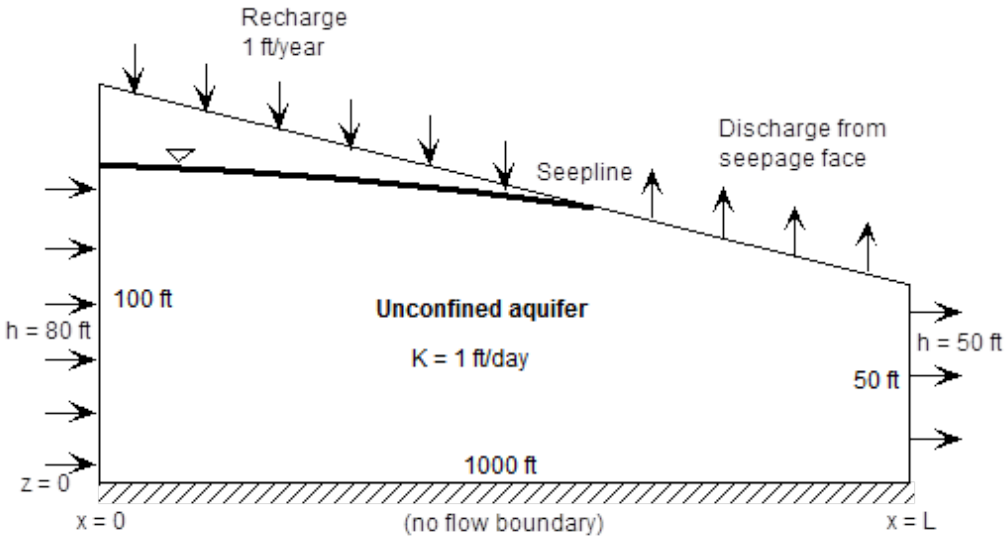

Figure 5.4.5: Schematic illustration of an unconfined aquifer experiencing both recharge and drainage at the ground surface; the seepline is unknown *a priori*.

There is a known analytical solution to this problem and the results of that solution are used as the observed values when creating the PEST model. The parameter being adjusted is the hydraulic

conductivity, K_x. The analytical solution is based on a value of 1.0 for this parameter. In this test problem, we set the K_x value to 0.9, to see how PEST does at estimating the correct value.

Table 5.4.4 shows the results of PEST testing. As can be seen, with only one parameter value to estimate, the parameter value estimated by the different modes of operation are identical to each other, $K_x = 0.980$. The ϕ value is also identical ($\phi = 1.6536 \times 10^{-2}$). When the ϕ value is divided by the number of observations (21) and square-rooted becomes the $RMSD = 0.028$, which is better than the RMSD reported in (Whiteside, 2016), $RMSD = 0.050$.

Because the PEST estimated value (0.98) is within the tolerance defined within the PEST control file (1×10^{-3}), PEST meets the criteria established in this test.

Table 5.4.4: Showing the mode of PEST operation, the parameter name, and the parameter value. The phi parameter is the sum of the squared weighted residuals.

Mode	Parameter	Value
hb	phi	1.6536E-02
sb	phi	1.6536E-02
s	phi	1.6536E-02
sp	phi	1.6536E-02
hb	kx	0.9804200000000000
sb	kx	0.9804200000000000
s	kx	0.9804200000000000
sp	kx	0.9804200000000000

5.5 Installation and Acceptance

To install PEST, the code must be compiled according to the instructions in the PEST manual and website and installed by the SRNL HPC personnel and located in the SRNL HPC apps directory. This will provide configuration change control, as this is a read-only directory only write-accessible by authorized personnel. See Listing 8 for a script to compile PEST.

Once compiled and installed in the appropriate location, all of the tests described in the Section 5.4 must be successfully executed from the production machines (including anolis, skinks, and/or compute nodes) in order for PEST to be accepted. The results from these tests will be compiled into a report similar to Section 5.4. Should a new version of PEST be installed, the test cases will be run and a new test report generated, as described in the Section 5.6.

5.5.1 User Instructions

The approved operating system for PEST is the GNU/Linux operating system.

The user's interaction with the software will primarily be through a shell script, an example of one that sends model runs to the SRNL HPC is included in Listing 9.

The necessary training for a user includes how to write PEST control files and edit the model template files. Full instructions and examples are found in the PEST manual (Doherty, 2016a) and can also be seen in the test case code.

Input and Output formats for PEST are simple text files, a full description of the input and output specifications are found in the PEST manual (Doherty, 2016a).

Because parameter estimation and convergence is dependent on the model, the limitation of PEST is dependent on the complexity of the model code and its execution time. This limit will need to be determined by the user.

A description of error messages and PEST's response can be found in the PEST manual (Doherty, 2016a). If further support is needed, please see Section 6.3 for proper handling of those issues as well as obtaining additional user and maintenance support.

Installation will be done by SRNL HPC staff. Available sample/examples of execution are found in Section 5.4.

To test PEST, you may execute the test cases described in Section 5.4.

There are no end-user specifications regarding cyber security or risk and safety beyond understanding PEST's limitations, as described in Section 5.1.

5.5.2 Review and Approval

The Design Agency shall assure the installed version of PEST reproduces the test cases described in Section 5.4.

5.6 Operation and Maintenance

The active use of PEST will follow the instructions provided within the manuals (Doherty, 2016a,b).

Should problems or errors within PEST be discovered during this phase, they will be handled according to Section 6.3. If this action results in software modifications, those modifications shall be approved, documented, verified and validated, and controlled in accordance with the appropriate Life Cycle Plans. This will include creating new test cases to document the problem. After the software is modified these new and previous test cases will be run to ensure regressions are not introduced into the software.

When the software is run in a new computer environment, the test cases will be run to ensure all software dependencies are available. Periodic in-use tests are not necessary for PEST, as the computing environment will not affect required performance.

5.7 Retirement

Because PEST is existing software, retiring PEST from service shall follow these steps:

1. The CTF will notify other users that PEST is being retired
2. There are no Cyber-Security implications for retiring PEST
3. Retiring PEST implies either there is software available to replace its functions or that the functions provided by PEST are no longer needed. In this situation, there are no future Risk and/or Safety issues with retiring PEST. After PEST is retired, the only Risk and/or Safety implications would be if there is a need to re-run models that used PEST. In this case PEST could either be "unretired" or a case could be made for using the current methods to examine those models.
4. Delete PEST and it's supporting software from the installation location, likely the SRNL HPC servers
5. The CTF will change the status of the SWCD Documentation to retired

6 SQA Actions

The Configuration Control and Evaluation of the software are required under Level "C" classification and the Problem Reporting and Corrective Action and the Risk and Safety Analysis components will be done via a graded approach.

6.1 Configuration Control

The method used to control, uniquely identify, describe, and document the configuration of each version or update of a computer program and its related documentation shall be accomplished as described within the Installation and Acceptance (Section 5.5) and Operation and Maintenance (Section 5.6) portions of the Life Cycle Plans (Section 5).

6.2 Evaluation

The evaluation of PEST per Manual 1Q Procedure 20-1 section 5.6 is documented within this SQAP. The following requirements are documented within the referenced sections:

- Determine the classification - Section 2
- Determine the adequacy of software documentation to support testing, operation, and maintenance - Sections 5.4 and 5.6
- Identify activities to be performed throughout the applicable life cycle of the software including preparation of required documentation and performance of required reviews and/or tests - Section 5
- Determine the software's capabilities and limitations for intended use - Section 5.4
- Specify test plans and test cases required to validate the capabilities within the stated limitations - Section 5.4
- Identify instructions for software use within the limits of its capabilities - Section 5.6
- Identify any exceptions to the life cycle documentation and its justification - Section 5
- Identify adequacy of cyber security requirements for use at SRS - As appropriate within each Life Cycle subsection
- Identify adequacy of the risk and safety requirements for use at SRS - As appropriate within each Life Cycle subsection
- The acquired software shall be identified and controlled during the dedication process - Section 5.5 and Section 6.1

6.3 Problem Reporting and Corrective Action

If a problem (bug) is discovered in PEST, it will be reported to the CTF. The CTF will submit it to the PEST program author, John Doherty, at pestsupport@ozemail.com.au.

In addition, the CTF and designated users shall report software problems/issues to the Performing Manager. The Performing Manager will notify recipients of information generated by PEST, and with input from the CTF and information recipients, shall determine appropriate corrective action. All software problem reporting and corrective actions, where applicable, shall comply with the requirements of Manual 1Q, QAP 20-1 Section 5.8. Problem reporting and corrective action for other potential business programs will be controlled by the respective Performing Manager.

7 References

Doherty, J. (2016a). *PEST Model-Independent Parameter Estimation User Manual Part I: PEST, SENSAN and Global Optimisers*. Watermark Numerical Computing, 6th edition.

Doherty, J. (2016b). *PEST Model-Independent Parameter Estimation User Manual Part II: PEST Utility Support Software*. Watermark Numerical Computing, 6th edition.

Whiteside, T. (2016). PORFLOW testing and verification document. Technical Report SRNL-TR-2016-00012, Rev. 0., Savannah River National Laboratory, Savannah River Site, Aiken SC.

Appendix A Code Listings

Listing 8: compile_pest.sh

```
#!/bin/bash

make clean
make cppp
make -f pest.mak all
make clean
make -f ppest.mak all
make clean
make -f pestutl1.mak all
make clean
make -f pestutl2.mak all
make clean
make -f pestutl3.mak all
make clean
make -f pestutl4.mak all
make clean
make -f pestutl5.mak all
make clean
make -f pestutl6.mak all
make clean
make -f sensan.mak all
make clean

echo "type this: \n make    f beopest.mak all"
echo "type this: \n make clean"
echo "type this: \n make install"
```

Listing 9: run_hpc_beopest.sh

```
#!/bin/bash

PWD=`pwd`
BEOPEST=${PWD}/bin/beopest
HOSTNAME=`hostname`
HPCSUBMIT=/hpc/home/users/submitjob.script

echo ${BEOPEST}

MODEL="model.sh" #I picked this name - maybe better way could be found"

PST="*.pst"

MPATHS=(`find testcases -maxdepth 2 -name ${PST}`)

echo "Pick which QA case to run (digit), followed by [ENTER]:"

I=0
for MPATH in ${MPATHS[@]}; do
echo "${I}) ${MPATH}"
I=$((I+1))
done

read QA_CASE_NUM

QA_CASE=${MPATHS[QA_CASE_NUM]}

printf "How many slaves do you want? (1-9), followed by [ENTER]:\n"
read NUMSLAVES

printf "Running ${QA_CASE}\n"

DIR=${PWD}/$(dirname "${QA_CASE}")
PST=$(basename "${QA_CASE}")

PST=${PST%.pst}

#change to the actual test case directory
cd ${DIR}
```

```
#skip making a RMF file
if false; then
printf "creating RMF file\n"
PARLAM=-3
CWAIT=.2
RUNTIME=5
RMFFILE=""
RMFFILE="${RMFFILE}prf\n"
RMFFILE="${RMFFILE}${NUMSLAVES} 0 ${CWAIT} ${PARLAM}\n"
for ((I=1;I<=${NUMSLAVES};I++)); do
RMFFILE="${RMFFILE}slave_${I} ./slave_${PST}_${I}\n"
done
RTIMELINE=""
for ((I=1;I<=${NUMSLAVES};I++)); do
RTIMELINE="${RTIMELINE}${RUNTIME} "
done
RTIMELINE="${RTIMELINE}\n"
RMFFILE="${RMFFILE}${RTIMELINE}"
printf "${RMFFILE}" > ${PST}.rmf
fi
#end of coment out

pwd

rm screenlog.0
rm stderr.txt

printf "Starting master...\n"
screen -A -d -m -L -S master
RUNNING=""
SLEEP=1
PORT=0
read lowerPort upperPort < /proc/sys/net/ipv4/ip_local_port_range
for (( PORT = lowerPort; PORT <= upperPort; PORT++ )); do
echo "trying ${PORT}"
screen -S master -p 0 -X stuff "${BEOPEST} ${PST} /h :${PORT} 2>stderr.txt$(pri
sleep ${SLEEP}
RUNNING=`wc -l stderr.txt | cut -d ' ' -f 1`
if [ ${RUNNING} -eq 0 ]; then
break
```

```
fi
done
echo "master is running"

SLAVEFILE="slave.sh"
rm ${SLAVEFILE}
JOB=`echo "${SLAVEFILE}" | cut -d. -f1`

printf "creating slave file\n"
SLAVE=""
SLAVE="${SLAVE}\#!/bin/bash\n"
SLAVE="${SLAVE}${BEOPEST} ${PST} /h ${HOSTNAME}:${PORT}\n"
printf "${SLAVE}" > ${SLAVEFILE}
chmod +x ${SLAVEFILE}

printf "Starting slaves...\n"
for ((I=1;I<=${NUMSLAVES};I++)); do
SDIR="slave_${I}"
mkdir ${SDIR}
cd ${SDIR}
cp ../* .

echo "\r" | ${HPCSUBMIT} ${SLAVEFILE} ${JOB} 10

cd ..
done
RUNNING=""
SLEEP=5

#wait until done (Speedup is last line printed out by beopest master)
until [ ${#RUNNING} -gt 0 ]; do
printf "sleeping for ${SLEEP} seconds\n"
sleep ${SLEEP}
RUNNING=`tail -2 screenlog.0 | grep "Speedup"`
done


#get beopest to hang up the port...
#send return to quit beopest?-maybe (doesn't seem to work)
screen -S master -p 0 -X stuff "$(printf \\r)"
screen -S master -X quit
```

```
#make beopest hang up the port
fuser -k -n tcp ${PORT}

rm screenlog.0
rm stderr.txt

cd ${PWD}

exit
```

Listing 10: run_qa.sh

```
#!/bin/bash


get_param_val(){
for TYPE in ${!TYPES[@]}; do #type of PEST to run
cd ${TYPE}
VAL=`grep ${PARAM} *.par | cut -d' ' -f17 | tr -d '\n'`
#     echo ${PARAM} ${VAL}
#     echo ${VAL}
PARAMTMP="${PARAMTMP}${TYPE}\t${PARAM}\t${VAL}\n"
cd ..
done
PARAMTMP="${PARAMTMP}----------\n"
}

#---------- PARALLEL PEST

make_rmf(){
printf "creating RMF file\n"
local PARLAM=0   #set the parallel-zation of the Marquardt lambda search to off
local CWAIT=.2
local RUNTIME=5
local RTIMELINE=""
local RMFFILE=""

RMFFILE="${RMFFILE}prf\n"
RMFFILE="${RMFFILE}${NUMSLAVES} 0 ${CWAIT} ${PARLAM}\n"
for ((I=1;I<=${NUMSLAVES};I++)); do
RMFFILE="${RMFFILE}slave_${I} ./slave_${I}\n"
done
for ((I=1;I<=${NUMSLAVES};I++)); do
RTIMELINE="${RTIMELINE}${RUNTIME} "
done
RTIMELINE="${RTIMELINE}\n"
RMFFILE="${RMFFILE}${RTIMELINE}"
printf "${RMFFILE}" > ${PST}.rmf
}
```

```
make_slaves(){
printf "Making slavedirs and starting slaves\n"
for ((I=1;I<=${NUMSLAVES};I++)); do
SDIR="slave_${I}"
mkdir ${SDIR}
cp * ${SDIR}/.
cd ${SDIR}
screen -A -d -m -S "${SDIR}" "${PARASLAVE}"
screen -S "${SDIR}" -p 0 -X stuff "./${MODEL}$(printf \\r)"
cd ..
done
}

kill_slaves(){
printf "Killing slaves\n"
for ((I=1;I<=${NUMSLAVES};I++)); do
SDIR="slave_${I}"
screen -S "${SDIR}" -X quit
rm -rf ${SDIR}
done
}


#--------------- BEOPEST

start_master(){

#cleanup old stuff if it's around
rm screenlog.0
rm stderr.txt

printf "Starting master...\n"
screen -A -d -m -L -S master
RUNNING=""
SLEEP=1
PORT=0
read lowerPort upperPort < /proc/sys/net/ipv4/ip_local_port_range
for (( PORT = lowerPort; PORT <= upperPort; PORT++ )); do
echo "trying ${PORT}"
screen -S master -p 0 -X stuff "${BEOPEST} ${PST} /h :${PORT} 2>stderr.txt$(pri
sleep ${SLEEP}
RUNNING=`wc -l stderr.txt | cut -d' ' -f1`
```

```
if [ ${RUNNING} -eq 0 ]; then
break
fi
done
echo "master is running"
}

make_slavefile(){
SLAVEFILE="slave.sh"
rm ${SLAVEFILE}
JOB=`echo "${SLAVEFILE}" | cut -d. -f1`

printf "creating slave file\n"
local SLAVE=""
#     SLAVE="${SLAVE}\#!/bin/bash\n"
SLAVE="${SLAVE}${BEOPEST} ${PST} /h ${HOSTNAME}:${PORT}\n"
printf "${SLAVE}" > ${SLAVEFILE}
chmod +x ${SLAVEFILE}
}

start_beopest_slaves(){

printf "Starting slaves...\n"
for ((I=1;I<=${NUMSLAVES};I++)); do
SDIR="slave_${I}"
mkdir ${SDIR}
cd ${SDIR}
cp ../* .

if [[ ${TYPE} == "sb" ]]; then
screen -A -d -m -S ${SDIR}
screen -S "${SDIR}" -p 0 -X stuff "${BEOPEST} ${PST} /h ${HOSTNAME}:${PORT}$(pr

elif [[ ${TYPE} == "hb" ]]; then
printf "\r" | ${HPCSUBMIT} ${SLAVEFILE} ${JOB} 10
fi

cd ..
done


}
```

```
wait_then_kill_all(){
local RUNNING=""
local SLEEP=5

#wait until done (Speedup is last line printed out by beopest master)
until [ ${#RUNNING} -gt 0 ]; do
printf "sleeping for ${SLEEP} seconds\n"
sleep ${SLEEP}
RUNNING=`tail -2 screenlog.0 | grep "Speedup"`
done

for ((I=1;I<=${NUMSLAVES};I++)); do
SDIR="slave_${I}"
screen -S "${SDIR}" -X quit   #this only needed for sb types, but whatever
rm -rf ${SDIR}
done

#get beopest to hang up the port...
#send return to quit beopest?-maybe (doesn't seem to work)
screen -S master -p 0 -X stuff "$(printf \\r)"
screen -S master -X quit

#make beopest hang up the port
fuser -k -n tcp ${PORT}

#cleanup
rm screenlog.0
rm stderr.txt
}

#-------------------------------------------------------------------------------

MODEL="model.sh" #I picked this name - maybe better way could be found"

PWD=`pwd`
PEST=${PWD}/bin/pest
PARAPEST=${PWD}/bin/ppest
PARASLAVE=${PWD}/bin/pslave
BEOPEST=${PWD}/bin/beopest
```

```
HOSTNAME=`hostname`
HPCSUBMIT=/hpc/home/users/submitjob.script

NUMSLAVES=3

declare -A TYPES
#TYPES["s"]=${PEST}
TYPES["sp"]=${PARAPEST}
#TYPES["sb"]=${BEOPEST}
#TYPES["hb"]=${BEOPEST}


#runPEST=false
runPEST=true

PARAMFILEFRONT="${PWD}/paramvals"

#TESTCASES
T1=${PWD}/testcases/t1 #twoline - PEST manual
T2=${PWD}/testcases/t2 #43      - PORFLOW QA
T3=${PWD}/testcases/t3 #442     - PORFLOW QA

T1PARAMS=(s1 s2 y1 xc)
T2PARAMS=(kxu kxc)
T3PARAMS=(kx)


#TESTCASES=(${T3} ${T2} ${T1})
TESTCASES=(${T3} ${T2} ${T1})


for TEST in ${TESTCASES[@]}; do #t1, t2, t3, ...
TNUM=`basename ${TEST}`
PHIVALS=${TNUM}  #t1
PARAMVALS=${TNUM}

PARAMFILE="${PARAMFILEFRONT}-${TNUM}.txt"
printf "" > ${PARAMFILE}

cd ${TEST}

for TYPE in ${!TYPES[@]}; do #type of PEST to run
```

```
echo "Doing test: ${TNUM} of type: ${TYPE}"

if [ ! -d ${TYPE} ]; then
mkdir ${TYPE}
fi
cp org/* ${TYPE}/.

cd ${TYPE}

PST=`ls *.pst`
PST=${PST%.pst}  #get the PEST control file

if ${runPEST}; then
EXE=${TYPES[${TYPE}]}

if [[ ${TYPE} == "s" ]]; then
echo "running ${PST} using ${EXE}"
${EXE} ${PST}

elif [[ ${TYPE} == "sp" ]]; then
make_rmf
make_slaves

echo "running ${PST} using ${EXE}"
${EXE} ${PST}

kill_slaves

elif [[ ${TYPE} == "sb" ]]; then
start_master
start_beopest_slaves
wait_then_kill_all

elif [[ ${TYPE} == "hb" ]]; then
start_master
make_slavefile
start_beopest_slaves
wait_then_kill_all
fi

else
echo "not running PEST"
```

```
fi

echo "getting Objective Function value"
csplit *.rec /Objective/ -s
VAL=`grep "Sum of squared" xx01 | cut -d' ' -f28`
PARAMTMP="${TYPE}\tphi\t${VAL}\n"
printf ${PARAMTMP} >> ${PARAMFILE}
rm xx*

cd ..
done #with this type of PEST

PARAMTMP="-----------------------------------------"
echo ${PARAMTMP} >> ${PARAMFILE}

PARAMTMP=""
echo "getting Parameter values"
if [[ ${TEST} == *"t1"* ]]; then
for PARAM in ${T1PARAMS[@]}; do
get_param_val
done
elif [[ ${TEST} == *"t2"* ]]; then
for PARAM in ${T2PARAMS[@]}; do
get_param_val
done
elif [[ ${TEST} == *"t3"* ]]; then
for PARAM in ${T3PARAMS[@]}; do
get_param_val
done
fi

cd ${PWD}

printf ${PARAMTMP} >> ${PARAMFILE}

done
exit;
```

Distribution:

S. E. Aleman, 735-A
B. T. Butcher, 773-42A
D. A. Crowley, 773-42A
G. P. Flach, 773-42A
L. L. Hamm, 735-A
C. G. Sherer, 773-41A
T. S. Whiteside, 773-42A
Env. Model. Files, 773-42A Rm 243
Records Administration (EDWS)