552391

Analysis Report for Task 7 of AP-114: Calibration of Culebra Transmissivity Fields

(AP-114: Analysis Plan for Evaluation and Recalibration of Culebra Transmissivity Fields, Revision 1)

Task Number: 1.4.1.1

10/30/2009

Authors:

David B. Hart — 11/12/09
David B. Hart, 6311
National Security Applications Department

Richard L. Beauheim — 11/12/09
Richard L. Beauheim, 6712
Repository Performance Department

Sean A. McKenna — 11/12/09
Sean A. McKenna, 6311
National Security Applications Department

Technical Review:

Kristopher L. Kuhlman — 11/12/09
Kristopher L. Kuhlman, 6712
Repository Performance Department

QA Review:

Mario J. Chavez — 11/12/09
Mario J. Chavez, 6710
Carlsbad Programs Group

Management Review:

Moo Lee — 11/18/09
Moo Lee, 6711
Performance Assessment and Decision Analysis Department

WIPP:1.4.1.1:TD:QA-L:RECERT:541153

[THIS PAGE INTENTIONALLY LEFT BLANK]

Table of Contents

List of Figures

List of Tables

[THIS PAGE INTENTIONALLY LEFT BLANK]

1 Introduction and Narrative Overview

This document records the activities done for Task 7 of AP-114 (Beauheim, 2008), "Analysis Plan for the Evaluation and Recalibration of Culebra Transmissivity Fields." This task is similar to Task 4 of AP-088 (McKenna & Hart, 2003). In this current task, the base transmissivity fields that were generated during Task 5 of AP-114 (Hart, Holt & McKenna, 2008) were calibrated against field data using an inverse parameter estimation method. Two hundred (200) stochastically generated "base" fields underwent calibration, and the 100 fields that had the best calibration were selected as the final results of this task.

This document provides a detailed description of the forward models, the field data used, the calibration process, and the results of the calibration. The narrative overview presented in this section calls out the location of the detailed descriptions within this document. Pre- and post-processing scripts were written for this task, the source codes for which are presented in Appendix G and which are qualified according to NP 9-1, Appendix C (see Chavez, 2008). The acquired and developed software programs that were used in the calibration process were qualified under NP 19-1 (Chavez, 2006), and they are:

- MODFLOW 2000 v1.6. Acquired; qualified under NP 19-1. (Harbaugh et al., 2000)
- PEST v9.11. Developed; qualified under NP 19-1. (Doherty, 2004)
    - Executables that are part of PEST: PPK2FAC, FAC2REAL, MOD2OBS, PEST, PPEST, PSLAVE, SVDAPREP, TEMPCHEK, PARREP, PICALC, PARCALC
- DTRKMF v1.00. Developed; qualified under NP 19-1.
- Utility scripts referenced in this text, displayed in Appendix G; qualified under NP 9-1 App. C
    - setup.sh, INIT2PEST, REAL2MOD, OBS2REAL, model.sh, runpest.sh, template_pmaster.sh, template_pslave.sh

The following commercial-off-the-shelf (COTS) software programs were used for generating the graphics and doing the routine calculations presented in this report; for calculations, the macros/files will be pointed out in Appendix F; for graphics, the data sources are called out and any reductions/transforms described, also in Appendix F:

- GnuPlot version 4.2
- Microsoft® Excel 2007
- Microsoft® PowerPoint 2007
- MATLAB® r2008b (v7.7.0)
- VarioWin 2
- Python 2.3

Task 7 was started by taking the base fields from the results of AP-114 Task 5 (Hart, Holt & McKenna, 2008), and selecting a set of pilot points to use for calibrating transmissivity (T). The initial values for T at these points were taken from the base fields. In addition to transmissivity, the horizontal anisotropy (A) for transmissivity, the storativity (S), and a section of recharge (R) were also calibrated. The same transmissivity zones described in Hart, Holt & McKenna (2008) were used for transmissivity in this task, and corresponding zones were used for anisotropy. Zones for storativity and recharge were based on other analyses completed in the area surrounding the WIPP site (see Section 3.1 – Creation of

Parameter Zones). Pilot points were selected for each parameter and initial values were selected that were consistent with the conceptual model prepared as part of Hart, Holt & McKenna (2008) (see Section 3.2 –Selection of Pilot Point Locations and Section 3.3 – Selection of Initial Values).

A model variogram for transmissivity was created using the field values calculated from pumping and other well tests. This variogram was also used for all parameters, as it was the only one that could be created from field data (see Section 3.4 – Creation of Transmissivity Variogram). This variogram was used as input to the PEST groundwater utility PPK2FAC, which creates algebraic kriging factors from a variogram, pilot point locations, and zonation data. Once generated, these factors can be used with the FAC2REAL PEST utility to take a set of parameter values and create a continuous field from point values (see Section 3.5 – Creation of Fields from Pilot Point Values). The T, A, S and R fields were then ready for use in the MODFLOW numerical model to produce simulated head and drawdown results (see Section 3.7 – MODFLOW Numerical Model).

Once the MODFLOW models produced simulated drawdown and head results, the modeled results were compared to the field data for the tests that were modeled. The *residual of an observation* is calculated in PEST as the weighted difference between measured and modeled data. The observation weights were selected in order to make the sum of the weighted steady-state head errors approximately equal to the sum of errors of four observation wells in a transient pumping test, provided that all error values were equal in magnitude. Because the weights are constant throughout, the weights provide a counterbalance to bring the calibration back into balance if one type of test (steady-state or transient) starts to overwhelm the calibration. In this document, as in the PEST documentation (Doherty, 2004), the term "observation" defines a field measurement, but it does not define the value of that measurement; an observation is a measurement at a specific time and location. The residuals were then used by PEST to construct a finite-difference approximation of the Jacobian matrix used to optimize the pilot point parameter values. The goal of the optimization is to minimize the objective function value, a measure of the model (see Section 3.6 – Observations and Residuals).

Because traditional construction of the Jacobian matrix requires a forward model plus either one or two forward model calls for each parameter value used in the calibration, using 1000-plus parameters, as was done in this task, would be impossible without additional efficiency in the optimization. The singular value decomposition (SVD) Assist component of the PEST software reduces the size of the Jacobian by creating "super-parameters" through SVD of the Jacobian created with the original parameters. The SVD process required initial calculation of a single, full Jacobian matrix, but then reduced the subsequent number of required forward calls by a factor of four to six. The result was that three calls to PEST were required to calibrate the fields – first, a single full Jacobian calculation, which required 1200+ forward model calls; second, an SVD calibration on the reduced parameter set that ran up to 50 iterations, requiring between 100 and 400 forward model calls per iteration; third, a final PEST run with the best parameter results to create the final fields corresponding to the best parameter values calculated during the SVD-assisted calibration (see Section 4.1 – PEST Calibration Process). Total calibration time for a single base field was approximately 7 days using 6 processors (1 master node and 5 slave nodes).

One complication to the process arose after approximately 140 fields had been calibrated. It was found that the steady-state head values that were being used in the calibration process were out of date and several of the values were incorrect by several meters. After discussing this problem, it was decided to allow the currently *running* fields to complete with the erroneous values – yielding 150 fields – and then correct the values in the input data. An additional 50 fields were started using the corrected heads (see Beauheim, 2009). To deal with the incorrect values, a limited calibration was performed on the *results* of the first 150 fields (see Section 4.2 – Calibrated Correction of Steady State Head Values). The same process that has been described was followed for the limited secondary calibration, but the initial parameter values were taken from the calibrated results, only the necessary pilot point locations near the updated steady-state head values were allowed to be changed, and the SVD portion of the PEST re-calibration was limited to 10 iterations. The end result was 200 fields, with 150 of these fields having undergone a secondary calibration to incorporate corrected field observation data. The impact of this change is discussed in Section 4.3 – Evaluation of Impact of Multiple Calibration Processes.

The end result of the calibration was 200 field sets (transmissivity, anisotropy, storativity and recharge) calibrated to one set of steady-state heads and nine transient, multi-observation well pumping tests. The 100 best calibrated fields (those with the smallest residuals) were selected as the final results from this task (see Section 4.4 – Selection of Best-Calibrated Fields). These fields were used to generate a water budget file for input into the DTRKMF particle tracking simulator. The resulting advective-only transport travel times were used to generate a cumulative distribution of travel times for the resulting fields (see Section 5.1 – Calculation of Advective Transport Travel Time CDF). In addition, several statistical analyses were performed on the fields themselves to generate average values and variances in the field results (see Section 5.2 – Statistical Analysis of Resulting T, A, and S Field).

The complete calibrations were performed under quality-assurance (QA) run control (RC) with inputs and outputs stored in a concurrent versions system (CVS) repository on a central server. The calibrations required approximately six months of continuous runtime on a total of 80 processors (see Section 2 – Run Control and Hardware Used). The final results were renamed to represent the base field they started from. The T, A, S, and R fields, the MODFLOW steady-state water budget, and the DTRKMF particle track results were placed in a single Results module in the CVS repository (see 2.1 - CVS Repository).

2 Run Control and Hardware Used

This analysis employed run-control measures throughout. These processes aided in traceability and reproducibility of the analysis and provided additional guarantees against any loss of data integrity due to network, hardware, or human error. This run control was accomplished by using two elements. The first was to use a Concurrent Version System (CVS) repository for all the files that were part of this task. The second was to use the `ReadScript` run-control program developed by Tom Kirchner (see Kirchner, 2008) to launch the calibrations.

2.1 CVS Repository

The CVS repository that was used for this task is located at `elo.sandia.gov:/data/CVSLIB/Tfields`. Input files were checked out of the repository at the start of each calibration, and the results were checked back in to the repository as part of the calibration process. The repository resided on a machine that had both internal and external (tape-drive) backups that were performed on a regular basis.

The structure of the repository is presented here:

- **`Inputs`** – This module contained all the static and pre-defined input files that were used by the calibration, as well as the scripts that were written specifically for this task.
- **`RunControl`** – This module contained the templates for the `ReadScript.py` input files that were then individualized for each field.
- **`Outputs`** – This module contained all outputs from field calibrations.
- **`Results`** – This module contains the final fields that were selected as results of this task, as well as summary files generated during the final analysis.

2.2 Hardware Systems Used

Two Linux clusters were used in the calibration process: the ALICE cluster in Carlsbad, NM and the TETHYS cluster in Albuquerque, NM. The ALICE cluster is a networked Linux cluster running 13 Intel® Xeon™ Dual-core processor nodes. The operating system is Rocks 4, which is based on RedHat Linux 4. The TETHYS cluster is a multi-processor type cluster with 8 single-core AMD Athlon 64 nodes, 19 dual-core AMD Athlon 64x2 nodes, and 6 Pentium IV dual core nodes. All machines run RedHat Linux 5. Both clusters use Sun's Grid Engine software to handle queuing and execution of jobs. Not all nodes were online the entire time meaning that there were only 80 processors in use at any one time. The developed software was qualified according to NP 19-1 (Chavez, 2006), and pre- and post-processing scripts were qualified according to NP 9-1, Appendix C (Chavez, 2008).

2.3 Run Control System

The run control software `ReadScript.py` (Kirchner, 2008) was used with a set of input files to execute the calibration runs. The input scripts were prepared by the analysts, but were run by a run master, Marissa Reno. The scripts used the template input files provided in Appendix G, replacing the "TFNUM" variable with the current field ID. The base field was then checked out of CVS (Hart, Holt & McKenna, 2008), from the directory "`:ext:alice.sandia.gov:/nfs/data/CVSLIB/AP114:Task5/Outputs/runs/subtask4`", the executables and other input files checked out from their

respective locations, and the calibration was started. Following calibration, all the OUTPUT type files specified in the input script were checked back in to CVS automatically. On the TETHYS cluster, the CVSROOT variable was modified to include the ":ext:alice.sandia.gov:" prefix to ensure that the remote repository was used for all files. This detail is recorded in all the log files that were produced for individual runs, and are stored in the RunControl and Outputs subdirectories. In this document, the notation "CVS://Repository::Module/Subdir/File" will be used, where CVS:// represents the /nfs/data/CVSLIB directory on ALICE.

Additional run control information for this task, with a full file trace-back and record of commands, is provided in Appendix H. The log files should contain complete records of execution, and are located in CVS://Tfields::RunControl/LogFiles/*/*.rtf

3 Subtask A – Setup and Configuration

This subtask comprised the setup and configuration processes that were the same for every base field that was calibrated. Subtask A included the creation and definition of zones for each of the parameters, the selection of pilot point locations and initial values, and initialization of the scripts and data files necessary to perform the calibration process. Because of the stochastic nature of the transmissivity fields, unique zones are associated with each field, as defined in Hart, Holt & McKenna (2008). The process to set up each field was the same, but certain elements, such as the exact number of pilot point locations used, were unique to each field.

The model domain and extent are identical to the domain defined in Hart, Holt & McKenna (2008) – a cell size of 100×100 meters, and a domain 28.4 km along the "X" (West to East) axis and 30.7 km along the "Y" (South to North) axis. The south-west node is centered at 601700 m Easting, 3566500 m Northing (UTM NAD27, zone 13). The base T fields were taken from the results of Hart, Holt & McKenna (2008). Some elements were created statically, and were used for every calibration. Some elements were created dynamically for each calibration, but used the same variables and parameters in their creation. The complete file creation process is shown in Figure 3-1, which outlines the execution of the INIT2PEST analysis script.

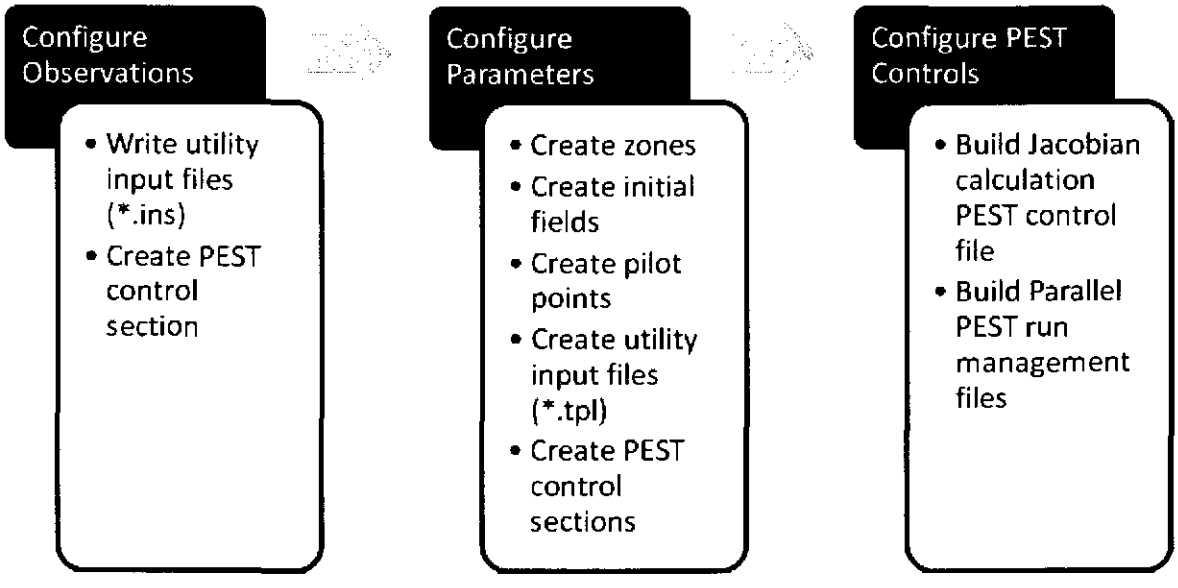


Figure 3-1. Process for setting up the initial PEST configuration and conditions for a T-field calibration. The files created by INIT2PEST are listed in Table F-2 in Appendix F.

3.1 Creation of Parameter Zones

Parameter zones were defined for each of the parameters to be calibrated. These zonations were defined to be consistent with the conceptual model of the Culebra flow defined in Hart, Holt & McKenna (2008). Figure 3-2 shows the different margins that define geologic zones and the location of the Culebra wells that have been drilled in the vicinity of the WIPP.

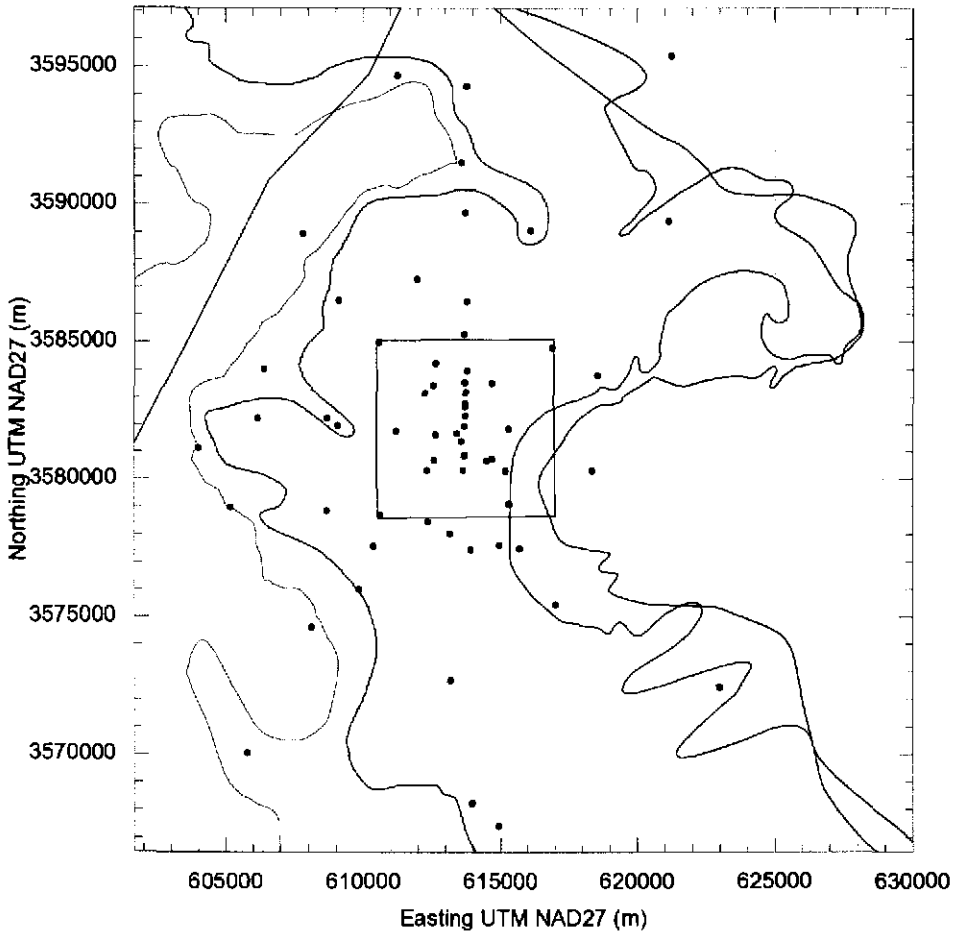


Figure 3-2. The model domain with margins and wells. From west to east along the center of the model, the margins are: No flow (black line on west), unconfined limit (orange), Salado dissolution (red), WIPP Land Withdrawal Boundary (LWB) (black square), M2/H2 Halite margin (green), M3/H3 Halite margin (blue).

3.1.1 Transmissivity Zones

The transmissivity zones were defined to be consistent with the zones that were produced in Hart, Holt & McKenna (2008). As shown in Figure 3-3, there was a high-transmissivity zone to the west (zone 2), corresponding to the area of Salado dissolution, and a mixture of higher and lower transmissivity values corresponding to the stochastic zones (zone 0 and 1) provided in the base fields that defined the center of the model domain. Unlike Task 5, where it was a separate zone, the area between the H2/M2 and H3/M3 margins was included in the same zone (zone 1) as the lower-T stochastic areas provided by the base fields. The area east of both the H2/M2 and H3/M3 margins – where the Culebra is bounded both above and below by halite-cemented elements – was defined to be its own zone (zone 3), as was done in Task 5. A no-flow boundary that roughly follows the center of Nash Draw is the final zone (zone 4) defined for transmissivity, and the only zone that applies to all parameters.

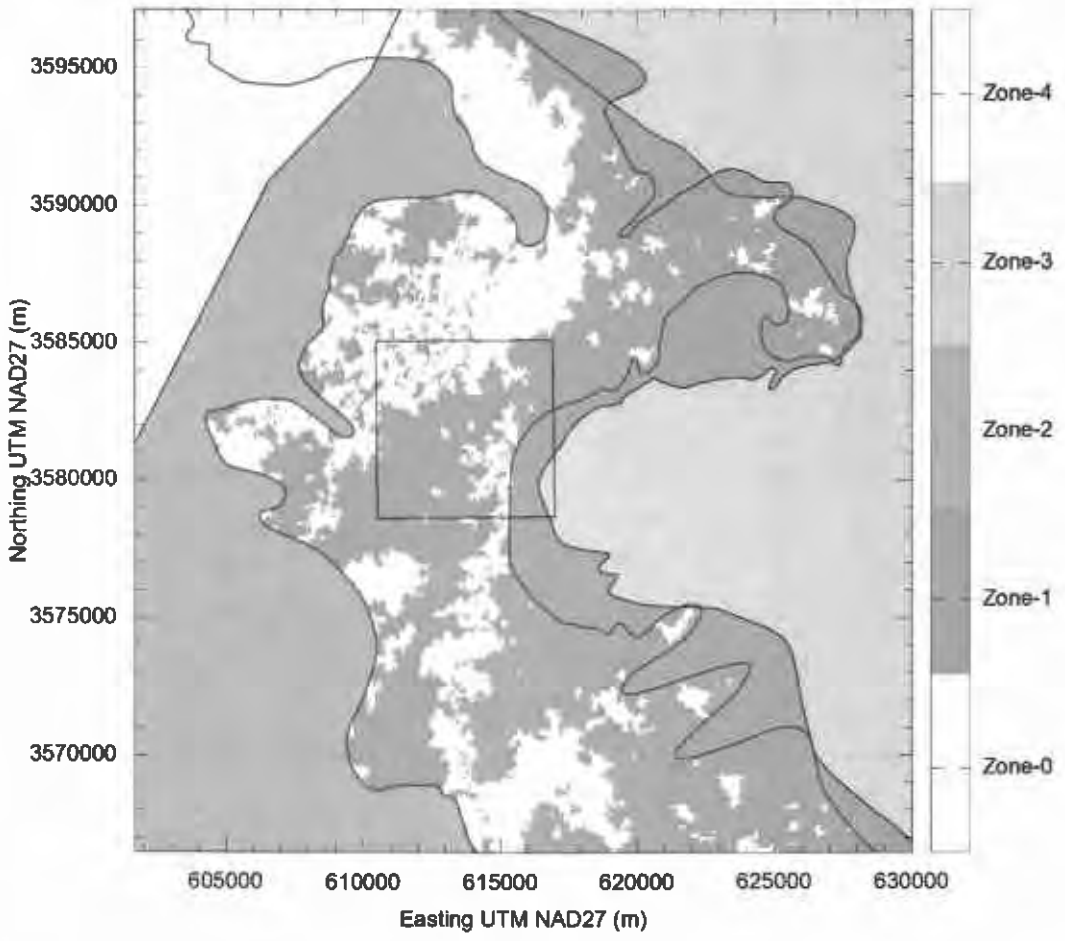


Figure 3-3. Example transmissivity zone map. Zones 0 and 1 are the stochastic zones created in Hart, Holt & McKenna (2008); Zone 2 is the high-T Salado dissolution area; Zone 3 is the very low-T halite-bounded area; Zone4 is no flow.

3.1.2 Anisotropy Zones

Because the anisotropy fields used in this task contain cell-by-cell anisotropy values for transmissivity (and only transmissivity), the exact same zones defined for transmissivity were reused as the zones for anisotropy. The transmissivity values in the North-South, or "Y", direction were calculated by multiplying the transmissivity value for the East-West, or "X", direction (given in the T field) by the anisotropy value (A) at a given cell:

$$T_{NS} = T_{EW} A$$

The map shown in Figure 3-3 represents the zonation for both A and T.

3.1.3 Storativity Zones

Besides the no-flow zone (zone 4), four zones were defined for storativity. The westernmost zone (zone 2) is the suspected unconfined zone, as described in Powers (2006). The largest zone (zone 0), with its western boundary roughly following the Salado dissolution margin, is the area considered to be fully confined. The area between these two zones (zone 1) is the transition zone, where there is some uncertainty as to whether the Culebra is fully confined. As with the T and A zones, the area east of both the H2/M2 and H3/M3 margins is a separate zone (zone 3), but in this case storativity is simply held constant at the initial confined-zone value. The main purpose in defining an "unconfined" zone as a storativity zone is to simplify the model by approximating the non-linear unconfined problem with a linear storativity model. By defining a much higher storativity value in the "unconfined" part of the domain, unconfined behavior can be approximately modeled using a confined – and more stable – numerical model. A map of the storativity zones is shown in Figure 3-4.

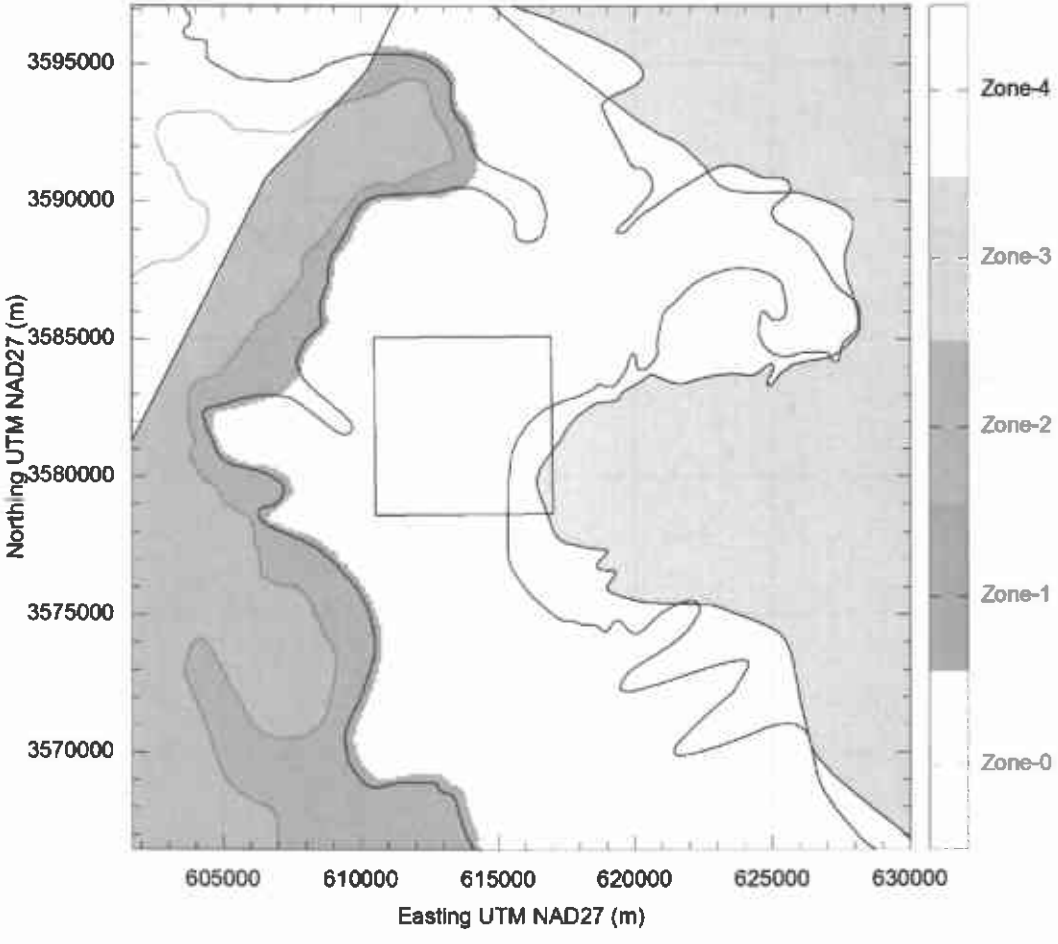


Figure 3-4. The storativity zones. Zone 0 is confined; Zone 1 is transition between confined and unconfined; Zone 2 is unconfined; Zone 3 is confined and held static at the initial confined value; Zone 4 is no flow

3.1.4 Recharge Zones

The conceptual model presented in Holt (1997) and in Hart, Holt & McKenna (2008) indicates that a groundwater divide exists somewhere southwest of the WIPP site. Previously (in McKenna & Hart, 2003), this groundwater divide was represented by extending the no-flow zone all the way to the southern boundary. Because the model used in this current task included the unconfined zone, it was decided to model the groundwater divide using recharge instead of a no-flow boundary. The areas of possible recharge were defined in AP-114 Task 1B (Powers, 2006). Recharge values had to be extremely small (on the order of 10^{-11} m/s) to allow MODFLOW to converge. Rather than try to determine which of the configurations presented in Task 1B was the "best" approximation – which was outside the scope of this task – a simple approximation that was similar to the older, no-flow approach was used. A recharge zone that was a simple line of cells extending NW to SE along the axis of the largest surface water feature (and roughly following the old no-flow boundary from McKenna & Hart, 2003) was used as the recharge zone; see Figure 3-5 for a map of the recharge zone.

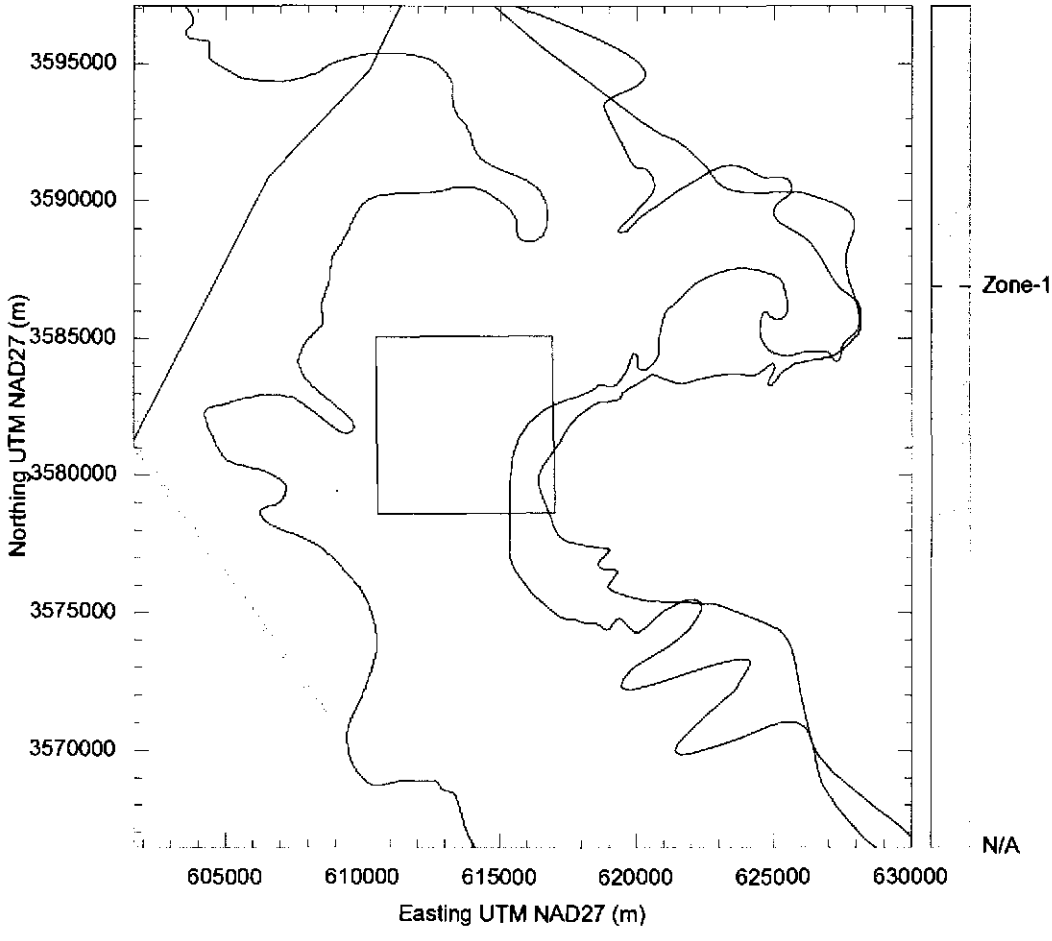


Figure 3-5. Zone 1 is the area of recharge; the remaining area was set to no recharge.

3.1.5 Flow, No-Flow and Fixed-Head Zones

While the boundary conditions were not variable parameters in the calibration, the definition of the specified head boundary conditions was an important part of the setup and configuration subtask. The no-flow zone in the northwest was defined to be the same as was used in AP-088. However, the significantly lower transmissivity in the far-eastern "halite-sandwiched" zone provided a more difficult boundary condition problem to solve. In the case of the western no-flow boundary, using "no flow" made conceptual sense, since here subsurface flow roughly followed the surface flow through Nash Draw – flow *should* parallel the no-flow boundary. In the east, this parallel result did not make sense conceptually or physically. Though the transmissivity in this area is extremely low (10^{-13} to 10^{-11} m^2/s), there should be some flow exiting along the zone margin, however minute. Testing at SNL-6 and SNL-15 indicates that the hydraulic heads in this area are at or above ground level (Roberts, 2007 and Bowman & Roberts, 2009). After several attempts to set fixed heads along the eastern boundary that would model this observation, it was decided to simply set the entire "halite-sandwiched" zone of extremely low T to fixed-head values, where the head was set to ground surface elevation. While this meant that the head values at SNL-6 and SNL-15 were no longer "estimable," it was the best way to model the nearly stationary nature of the water in this zone. The flow zones are shown in Figure 3-6, and the selection of the initial values is discussed in Section 3.3. The northern, western and southern flow boundaries were all fixed-head boundaries.

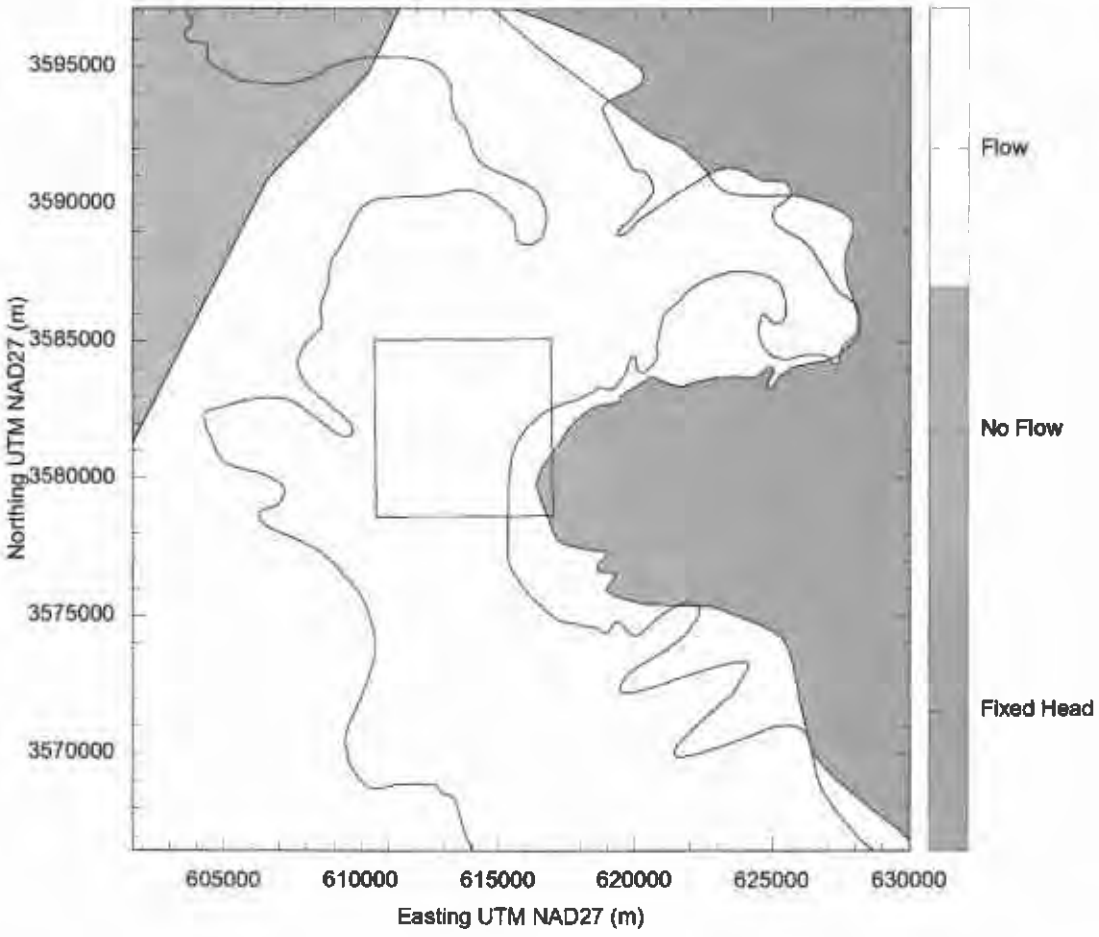


Figure 3-6. Flow zones: the fixed-head zone is green; the no-flow zone is salmon; the white area is normal flow. Note that the fixed-head zone includes one cell along the boundaries.

3.2 Selection of Pilot Point Locations

Once the zones were defined, pilot point locations were selected. There were two types of pilot points, fixed and variable, and two placement approaches, gridded and linear. Selection of the points for each parameter required a combination of both types and approaches. The algorithm for calculating placement was as follows:

1) Place fixed pilot points at locations of known parameter values,
    a. Define these points such that their influence extends only within their appropriate zone.
    b. Blank out an area surrounding each of these fixed values as "illegal" for additional pilot point placement.
2) Between each pair of wells (or locations) defined in a connectivity matrix listing the pairs of wells that were pumped and observed during pumping tests,
    a. Place pilot points 500 m away from the wells on a line that would connect those two wells.
    b. Place pilot points at equal distances along the remaining section of the connecting lines.
    c. If a pilot point is within a certain distance of a zone boundary, assign that point to both zones.
    d. Blank out the area around the new points as "illegal" for additional pilot point placement.
3) For each zone, place gridded pilot points
    a. Use a triangular grid pattern starting from the northwest corner.
    b. If a grid point falls within an "illegal" area, do not place the pilot point.
    c. If a grid point falls within a certain distance of a zone boundary, assign the point to both zones.

Because the zones are unique for each base field, this process was done for each field to be calibrated. The pilot point locations are the same for all fields, but some points may belong to two zones in one field and only to one zone in another. As a result, the pilot point identifiers are not necessarily the same between realizations.

3.2.1 Transmissivity Specific Pilot Point Settings

In addition to the pumping test wells, some extra points were used to place pilot points in the transmissivity fields. These were included along the northern and southern boundaries to try to limit the effects that the fixed-head boundaries would have on transient pumping and the steady-state model results. The measured transmissivity values from single-well pumping and slug tests were used as fixed transmissivity points at their corresponding wells – see Table 3-1 for the values used, and see Figure 3-7 for the locations of all the pilot points.

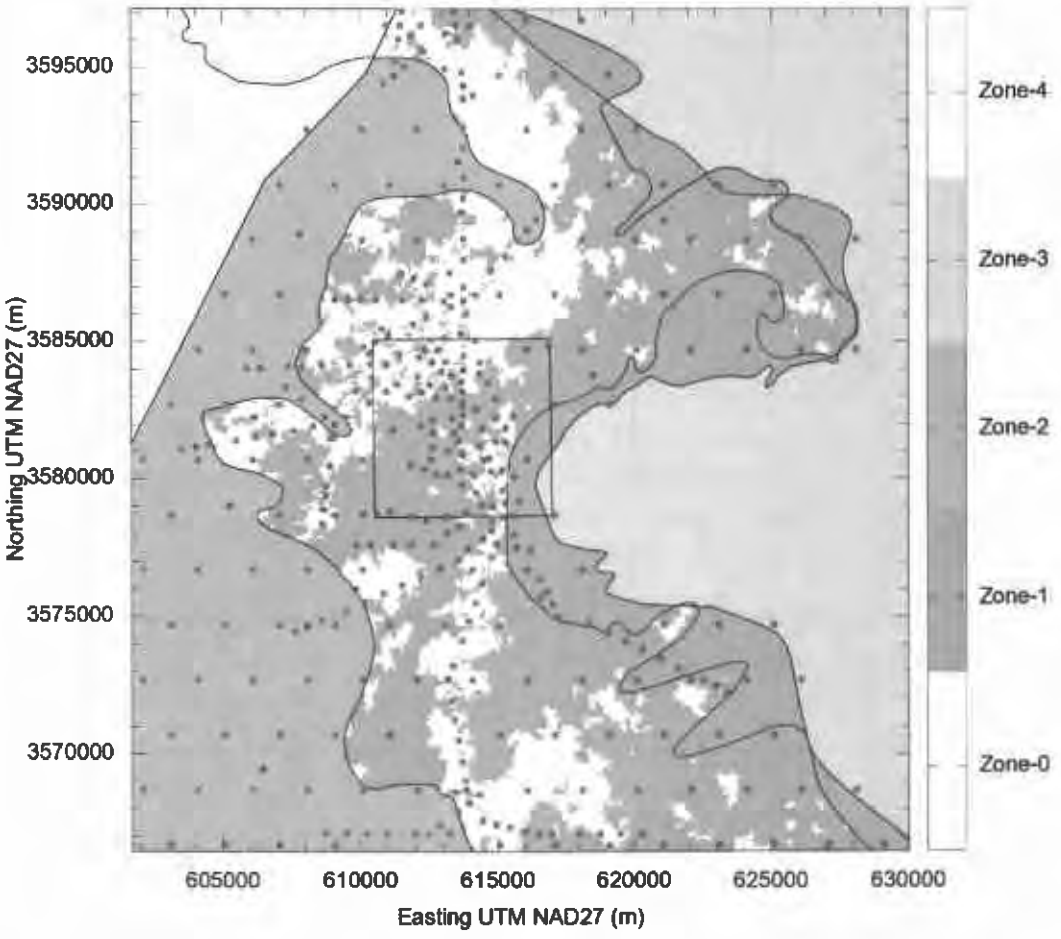


Figure 3-7. Transmissivity pilot point locations. Blue points are fixed values and red points are variable parameters. Points may exist in more than one zone, and those zones may change between realizations, but the pilot point locations are the same for all fields.

3.2.2 Anisotropy Specific Pilot Point Settings

Anisotropy was unique in that there were no fixed values and therefore no fixed pilot points. This result is due to the single-well tests not providing any estimate of anisotropy, and the multi-well tests providing too localized an estimate of anisotropy (only valid for a single cell or two in the model). See Figure 3-8 for the locations of anisotropy pilot points.

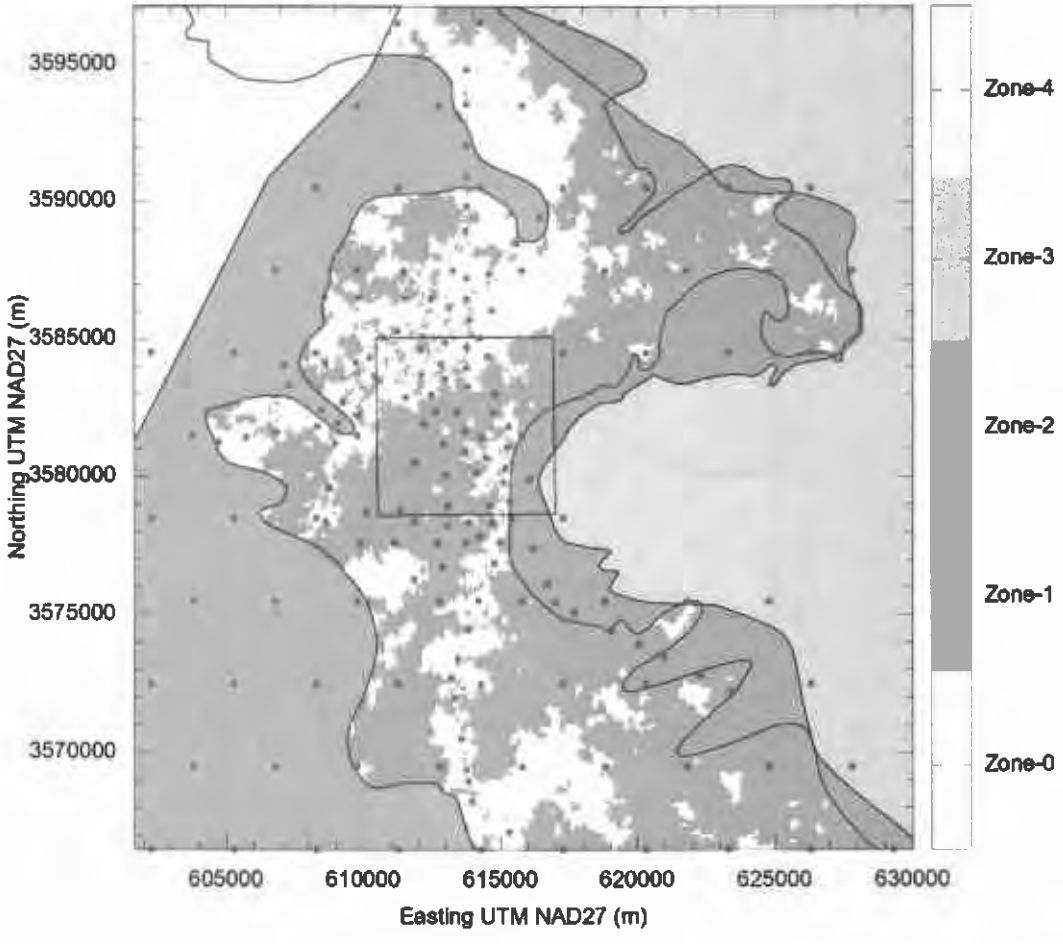


Figure 3-8. Anisotropy pilot point locations. All anisotropy pilot points were variable parameters.

3.2.3 Storativity Specific Pilot Point Settings

Storativity pilot points were placed in a seemingly opposite manner as for transmissivity, though the same algorithm was used. The only variable pilot points in the confined zone were along transient pumping test connectivity axes. The gridded points were set as fixed values, since there was no information that would allow effective calculation of storativity outside the transient tests. All pilot points located within the unconfined and transition zones were defined as variable. See Figure 3-9 for the location of storativity pilot points.

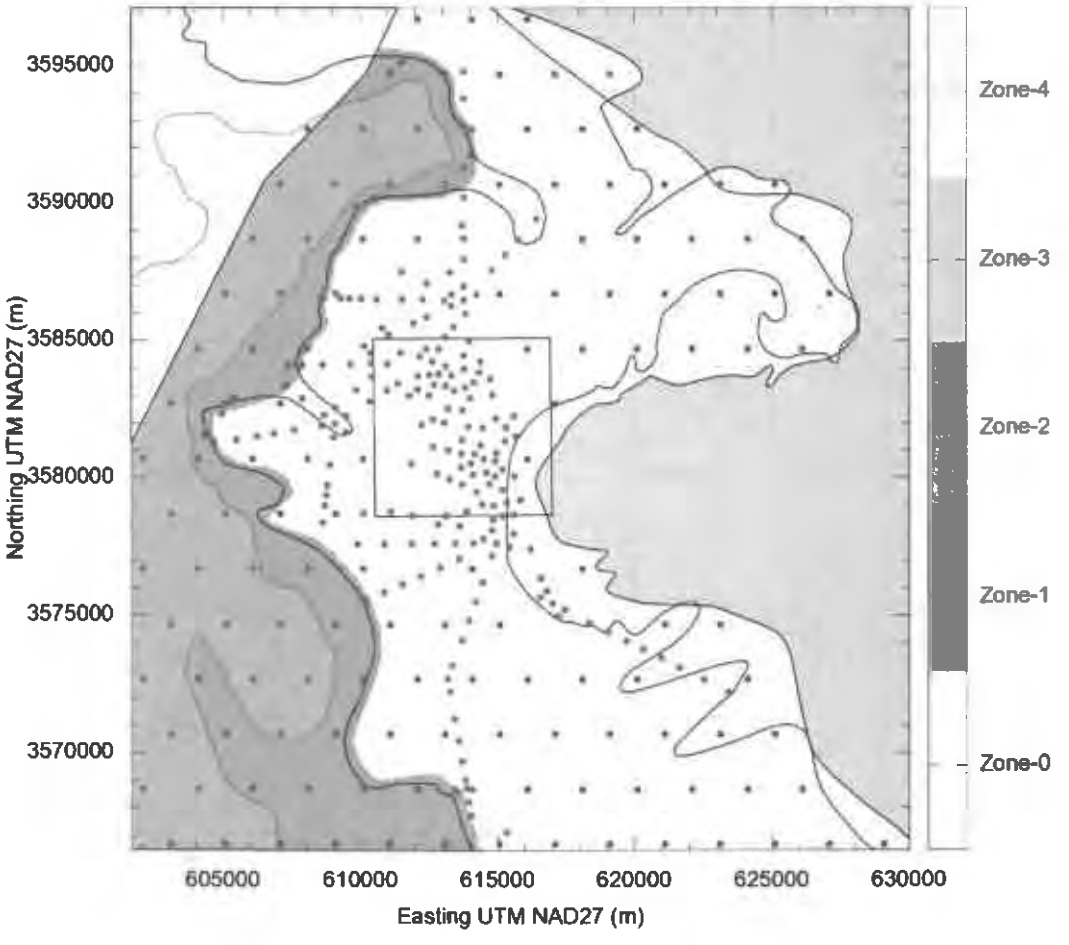


Figure 3-9. Location of storativity pilot points. Only pilot points along transient pumping test connections and points in the unconfined zones (zones 1 and 2) were variable, the remaining points were fixed at the initial storativity value.

3.2.4 Recharge Specific Pilot Point Settings

Because the recharge zone was a line, only four pilot points were needed in the entire zone. In this case, the pilot point nearest the western domain boundary was set as a fixed value of 10^{-30} m/s, which was interpreted as 0 by the pre-processors to MODFLOW, and the other three were variable. See Figure 3-10 for the location of the recharge pilot points.

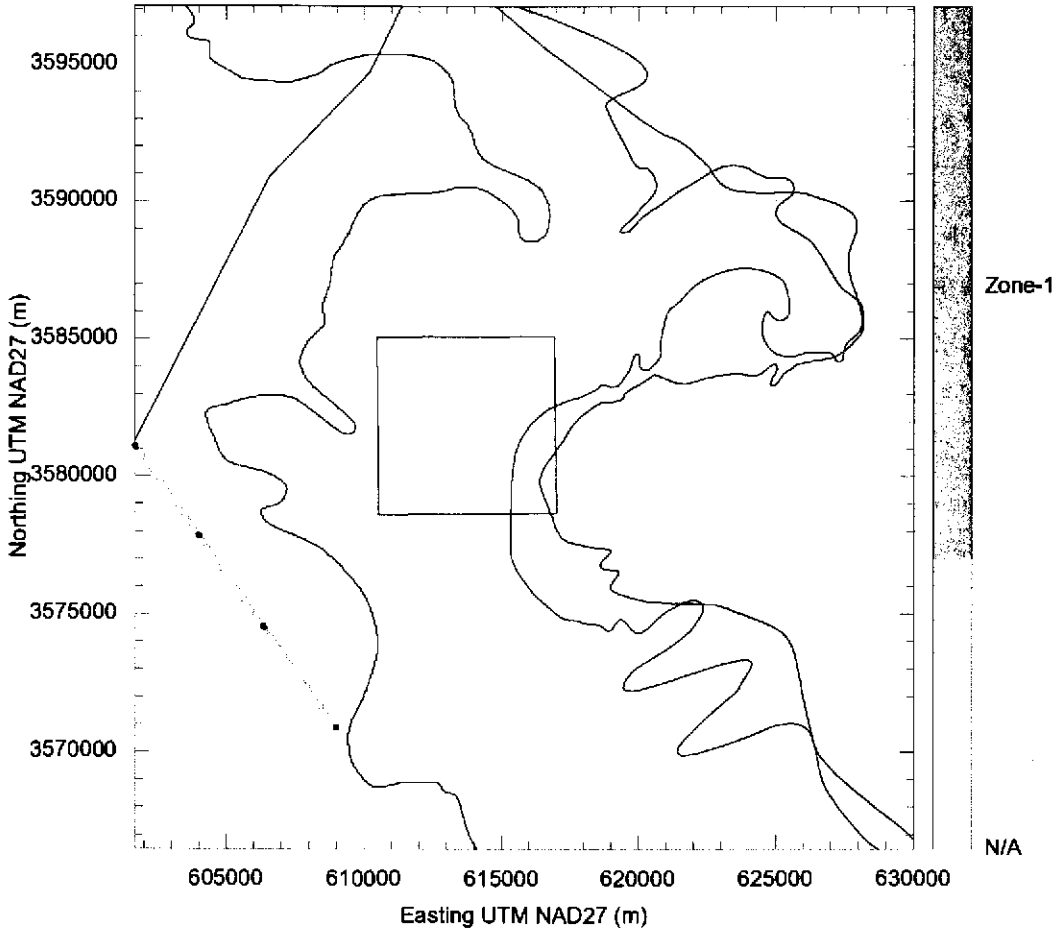


Figure 3-10. The location of the recharge pilot points.

3.3 Selection of Initial Values

3.3.1 Parameter Initial Values

The initial values for each of the pilot points were defined according to the conceptual model and the values presented in Hart, Holt & McKenna (2008). For transmissivity, this meant that the same equation used to create the base transmissivity fields was used to define the initial values for the pilot points, based on their zone. Anisotropy was set to isotropic conditions (A = 1.0) for all points. Storativity was defined to start at $10^{-5.0}$ for the confined zone (the same value that was used for storativity in AP-088), at $10^{-4.0}$ in the transition zone, and at $10^{-1.5}$ in the unconfined zone. Recharge was initialized as $10^{-11.0}$ m/s, which was a value that was found to be sufficiently small to allow MODFLOW to perform an initial run prior to PEST calibration. The zone-by-zone initial values for each parameter, and the limits placed on the range the values could take in calibration, are presented in Table 3-1.

Table 3-1. Initial values of parameters at pilot points. See Hart, Holt & McKenna (2008) for T values.

Parameter and Zone	$\log_{10}$ Value or Equation *($D_{x,y}$ is the depth to Culebra from ground surface)*	Calibration Limits on $\log_{10}$ Pilot Point Values
Transmissivity Zone 0	$-0.003484\ D_{x,y} - 3.6322$	[-19 , -1]
Transmissivity Zone 1	$-0.003484\ D_{x,y} - 5.6981$	[-19 , -1]
Transmissivity Zone 2	$-0.003484\ D_{x,y} - 2.9463$	[-19 , -1]
Transmissivity Zone 3	$-0.003484\ D_{x,y} - 10.4490$	[-19 , -1]
Anisotropy (All Zones)	0.0	[-0.5 , 0.5]
Storativity Zone 0	-5	[-5.5 , -4.5]
Storativity Zone 1	-4	[-6 , -0.5]
Storativity Zone 2	-1.5	[-2.5 -0.5]
Storativity Zone 3	-5	Fixed
Recharge Zone 1	-11	[-19 , -1]

3.3.2 Initial Head Field

The creation of the central initial head values was done using a multivariate equation based on the X and Y coordinates of the cell in question. The equation was designed to keep the northern boundary just above the measured head at SNL-1 and the southern boundary below the level measured at H-9c, and these constraints were the defining factors on the constants in the equation that follows. This process was done only once, and the result was used as a static input file for all calibrations. The creation of the field was done in MATLAB®, using the following equations:

$$X \stackrel{\text{def}}{=} -142{:}\ 1{:}\ 141$$
$$Y \stackrel{\text{def}}{=} -153{:}\ 1{:}\ 153$$
$$Z_{x,y} = 928 + 8\left(\left(\frac{y}{153}\right) + \text{sign}\left(\frac{y}{153}\right)\left(\text{abs}\left(\frac{y}{153}\right)\right)^{\frac{1}{2}}\right) + 1.2\left(\left(\frac{x}{142}\right)^{3} - \left(\frac{x}{142}\right)^{2} - \left(\frac{x}{142}\right)\right)$$

For values east of both the H2/M2 and H3/M3 boundaries, the ground-surface elevation was used as the initial head value. See Appendix A for the full calculation steps and process. The resulting initial head field is shown in Figure 3-11.

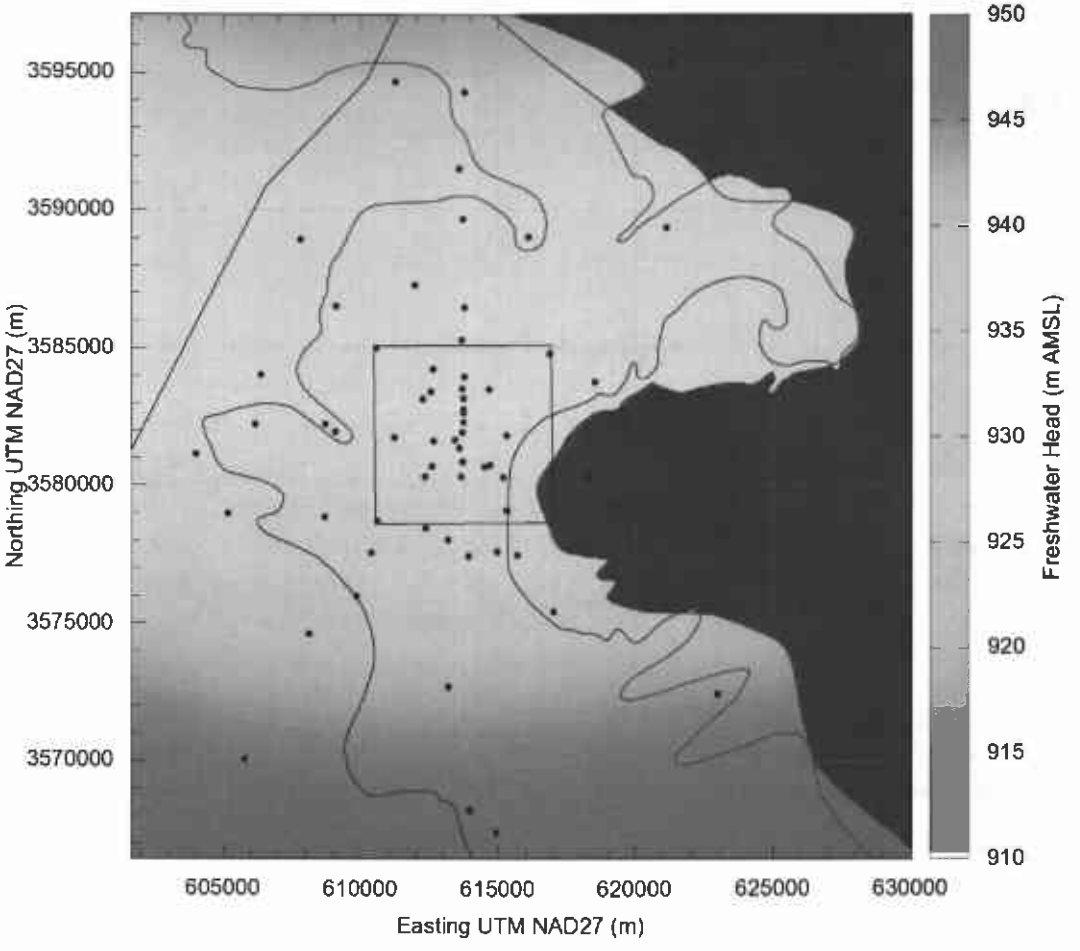


Figure 3-11. Initial head values for use in MODFLOW steady-state solution. Head values that are dark red were fixed at the ground surface elevation.

3.4 Creation of Transmissivity Variogram

The transmissivity variogram was created using transmissivity values calculated from well tests at 62 of the wells around the WIPP site. Wells outside the model domain and values at SNL-6 and SNL-15 were excluded from the calculation. The values at SNL-6 and SNL-15 are both several orders of magnitude lower than at the other wells, and are in a geologically distinct zone, which is the reason they were excluded. While initial calculations showed that there was some statistical anisotropy, there were not sufficient measurements in each individual zone to create an anisotropic variogram with confidence. The complete steps for creating the variogram are presented in Appendix B. The final parameters used are shown in Table 3-2.

Table 3-2. Parameters for parameter variograms, calculated from transmissivity using an omnidirectional variogram with lag spacing of 1500 m.

Parameter	Value
Model Type	Exponential
Nugget	0.02 $(\log_{10} T)^2$
Sill	1.95 $(\log_{10} T)^2$
Range	9500 meters

3.5 Creation of Fields from Pilot Point Values

The combination of the pilot point locations and values, the zone boundaries, and the controlling variogram, was used to create a full set of kriging factors for each parameter. The PEST utility program PPK2FAC was run once for each parameter to generate a factors file. The factors file contains a set of linear equations for every cell defining transmissivity (or anisotropy, storativity, or recharge) as a function of weighted pilot point values. This factor file, along with current pilot point values (modeled_points_*.dat), is used as input to the PEST utility FAC2REAL to produce a complete field of values from the smaller set of pilot point values. However, the T, S, A, and R pilot points are estimated in $\log_{10}$ space, meaning that they must be transformed back into real space prior to use in the numerical model. In addition to this, default values for non-kriged locations needed to be substituted for certain zones, and bulk shift values added to all values. To do this, the pre-processing script REAL2MOD was written; the script is documented in Appendix G.

3.6 Observations and Residuals

The observations for the numerical model were selected from the May 2007 steady-state head measurements at a selection of wells, and from the observations taken during pumping tests across the past 20 years. The total observation data points used are a subset of the possible observation data selected to provide accurate representation of the individual pumping test responses. The pumping tests used, and their references, are presented below.

- H-3 Pumping Test (Beauheim, 1987a)
    - 4 Observation Wells

    - 69 total observation data points
- WIPP-13 Pumping Test (Beauheim, 1987b)
    - 9 Observation Wells
    - 167 total observation data points
- H-11 Pumping Test (Beauheim, 1989)
    - 10 Observation Wells
    - 130 total observation data points
- P-14 Pumping Test (Beauheim & Ruskauff, 1998)
    - 5 Observation Wells
    - 69 total observation data points
- H-19/H-11 Multi-well Pumping Test (Beauheim & Ruskauff, 1998)
    - 7 Observation Wells
    - 143 total observation data points
- WQSP-1 Pumping Test (Beauheim & Ruskauff, 1998)
    - 2 Observation Wells
    - 36 total observation data points
- WQSP-2 Pumping Test (Beauheim & Ruskauff, 1998)
    - 4 Observation Wells
    - 77 total observation data points
- WIPP-11 Pumping Test (Toll & Johnson, 2006b) (Roberts, 2006)
    - 12 Observation Wells
    - 250 total observation data points
- SNL-14 Pumping Test (Toll & Johnson, 2006a) (Roberts, 2006)
    - 12 Observation Wells
    - 252 total observation data points
- Steady-State Head Measurements (Johnson, 2009) and see (Beauheim, 2009)
    - 44 Observation Wells
    - 44 total observation data points

3.7 MODFLOW Numerical Model

Inverse calibration requires a numerical model which produces results that can be compared to field or otherwise known information. In this task, a MODFLOW 2000 (Harbaugh et al., 2000) flow model was developed for the Culebra that could use the base fields generated in Hart, Holt & McKenna (2008) as inputs. As was done in McKenna & Hart (2003), the Link Algebraic Multi-Grid (AMG) (Mehl, 2001) solver was used to improve speed and performance. In addition to transmissivity, it was decided to calibrate the local anisotropy, storativity, and some recharge as parameters in the calibration. Having these four parameters – T, A, S, and R – required a slightly more complex MODFLOW model implementation than was used in AP-088. Specifically, both storativity and anisotropy were single values previously, and changing these to cell-by-cell values required the use of the Layer Property Flow (LPF) package instead of the Block Centered Flow (BCF) package used previously. Using recharge also required the addition of the recharge package. For the known information, steady-state heads from 2007 and drawdown results

from nine different pumping tests performed between 1985 and 2008 were used as the measured data. A diagram of the MODFLOW piece is shown in Figure 3-12.

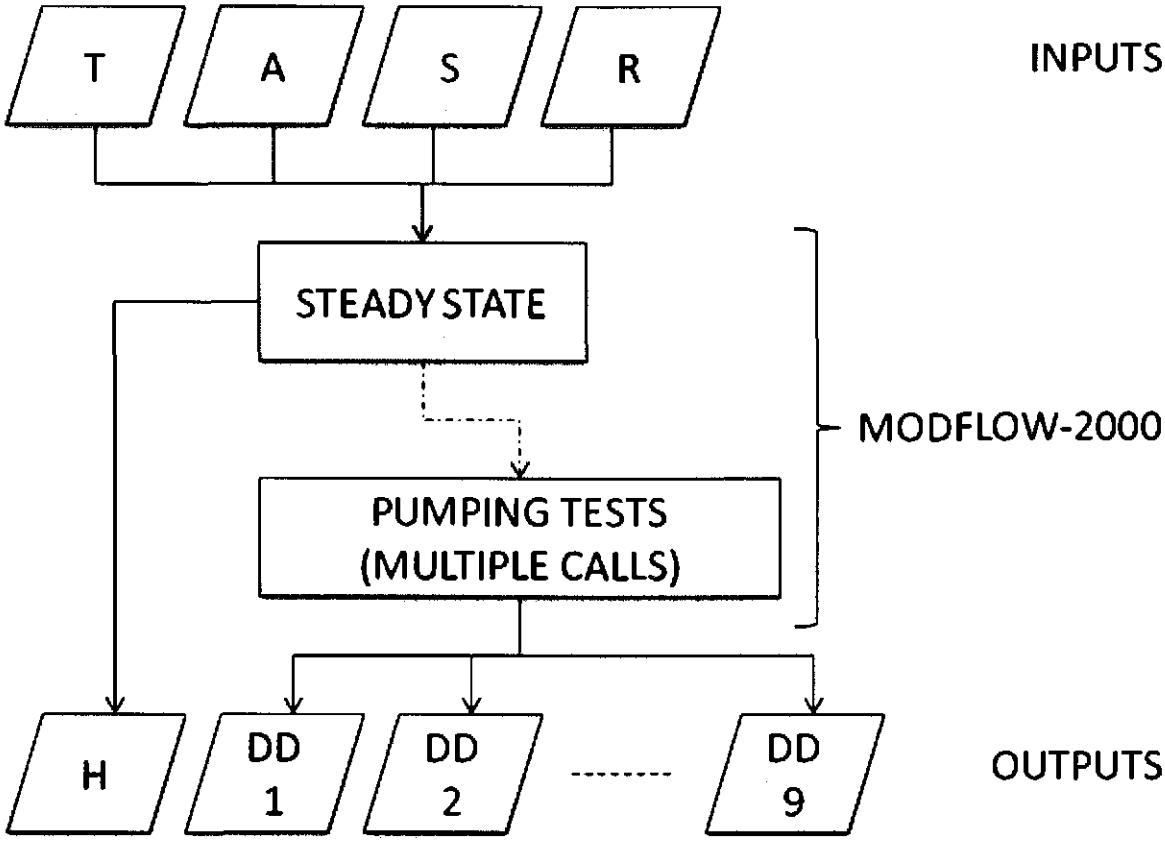


Figure 3-12. Flow chart showing the internal numerical model for the inverse calibration.

4 Subtask B – Inverse Calibration Process

The inverse calibration process used multiple forward calls to evaluate the impact that changing a parameter value had on the overall fit of the model to the *baseline* or field data (the objective function). This process was computationally intensive, and involved 80 processors on two different computing clusters running for six months to calibrate 200 fields.

4.1 PEST Calibration Process

The inverse calibration process was done using the PEST software and its groundwater utilities. One of the features in PEST version 9.11 is the Singular Value Decomposition (SVD) feature. This allows a large number of pilot point parameters to be reduced into a smaller set of "super parameters." The reduced number of super parameters can be optimized more quickly than optimizing all parameters individually, and the PEST process outputs the equivalent pilot point parameters at the end. It was this SVD feature that allowed the calibration process to be completed in six months using the 80 processors available. The PEST process is shown in Figure 4-1; the complete calibration process is shown in Figure 4-2.

The completed PEST "model" that was calibrated included the creation of the fields from the kriging factors and pilot points (PPK2FAC, FAC2REAL, REAL2MOD), the MODFLOW calls, and finally the observation extraction utilities (MOD2OBS and OBS2REAL) which extract modeled cell head or drawdown values from a binary MODFLOW output file. For SVD iterations, another preprocessor, PARCALC, is used to create the pilot point values from the super parameters. The model script, `model.sh`, the REAL2MOD script and the OBS2REAL script were written for this task, and are included in Appendix G. PPK2FAC, FAC2REAL, and MOD2OBS are part of the PEST program.

The first call to the PEST program was a single iteration to calculate the Jacobian matrix. This required one forward model call for each variable parameter value, of which there were approximately 1100. Once the Jacobian matrix was calculated, the SVDAPREP program decomposed the Jacobian into singular values and super parameters. The result was a set of 100 to 300 super parameters that were then used with a 50-iteration PEST calibration. The termination criteria were: a maximum of 50 iterations, 3 successive iterations without an improvement in the objective function, or a relative decrease of less than 0.001 in the objective function for 3 iterations.

Once termination criteria had been reached, the PEST program would output the best parameters to a file. This file was then used to create one final PEST control file which issued a single model run with the best parameters as input. The results of this final call were then used to calculate the measures of fit and the final fields.

The run control input file that ran the complete calibration process is presented in Appendix G. Using the `ReadScript.py` run control system allowed automatic check-in of the results to CVS following calibration.

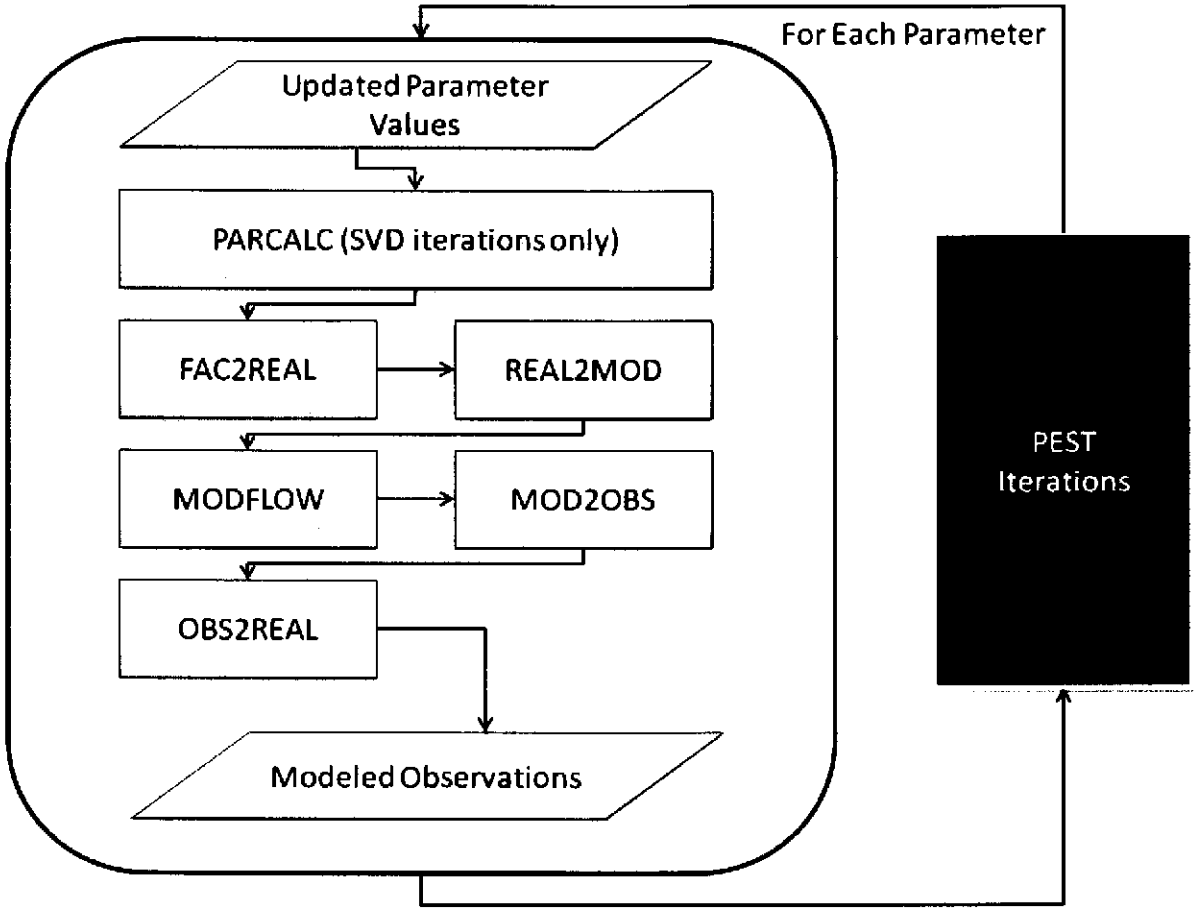


Figure 4-1. The PEST calibration process.

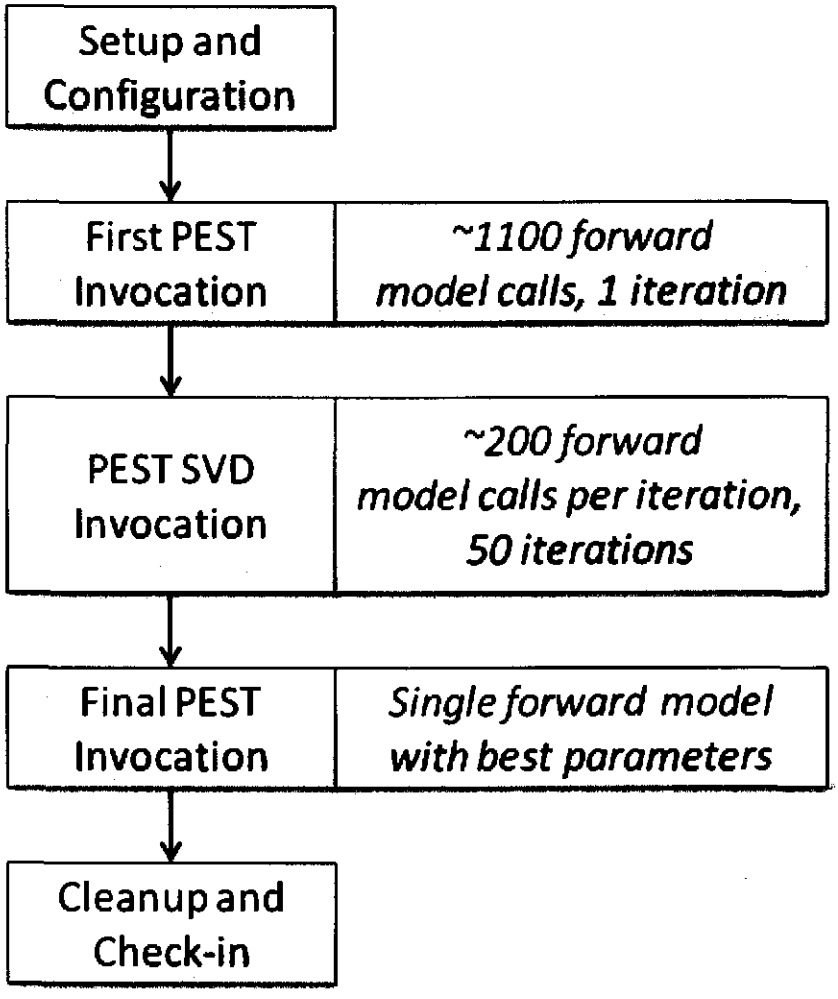


Figure 4-2. Complete calibration process for a single field. Steps 2-4 are contained in "pest_master.sh", which is an individualized copy of "CVS://Tfields::Inputs/scripts/template_pmaster.sh" created by "setup.sh".

4.2 Calibrated Correction of Steady State Head Values

Because some of the original steady-state data were found to be outdated during the calibration process, the fields that had been calibrated to the old data needed to be re-calibrated in some way to the new data. The two wells with the most significant changes, ERDA-9 and SNL-8, had more than one meter change from the old to new values. The ERDA-9 head was in some ways an expected change, because the calibrations had consistently been unable to match the old head value, which was too low compared to the higher corrected value. Without any re-calibration, correcting the value for ERDA-9 produced better model fits.

4.2.1 Localized Recalibration in the Vicinity of SNL-8

The head change at SNL-8 was a different matter altogether. The new value for SNL-8 – which was based on a recalculation of the freshwater head equivalent (Johnson, 2009) – was several meters lower than had been previously used in the calibration. Because SNL-8 was not used in any of the transient

calculations, and because it was to the east and up-gradient from the WIPP LWB, it was decided to try recalibrating only a section of the field to correct for the change in SNL-8. It was hoped that this would allow the T, S, A and R fields to change to match the SNL-8 head without requiring the week-long re-calibration for each of the affected fields if the entire domain was recalibrated.

The recalibration process involved fixing all the parameters that had previously been calibrated, except for those parameters in a rectangular area around and up-gradient from SNL-8. The complete area definition was 14 km east-west by 9 km north-south with the southwest corner at 616000 m Easting, 3580000 m Northing UTM NAD27 and is shown in red on Figure 4-3. All other aspects of the inverse calibration, including the forward model and the SVD assist process, were left the same. The resulting fields had significantly better fits to the steady-state heads, and little impact was seen on the transient test results (as was expected).

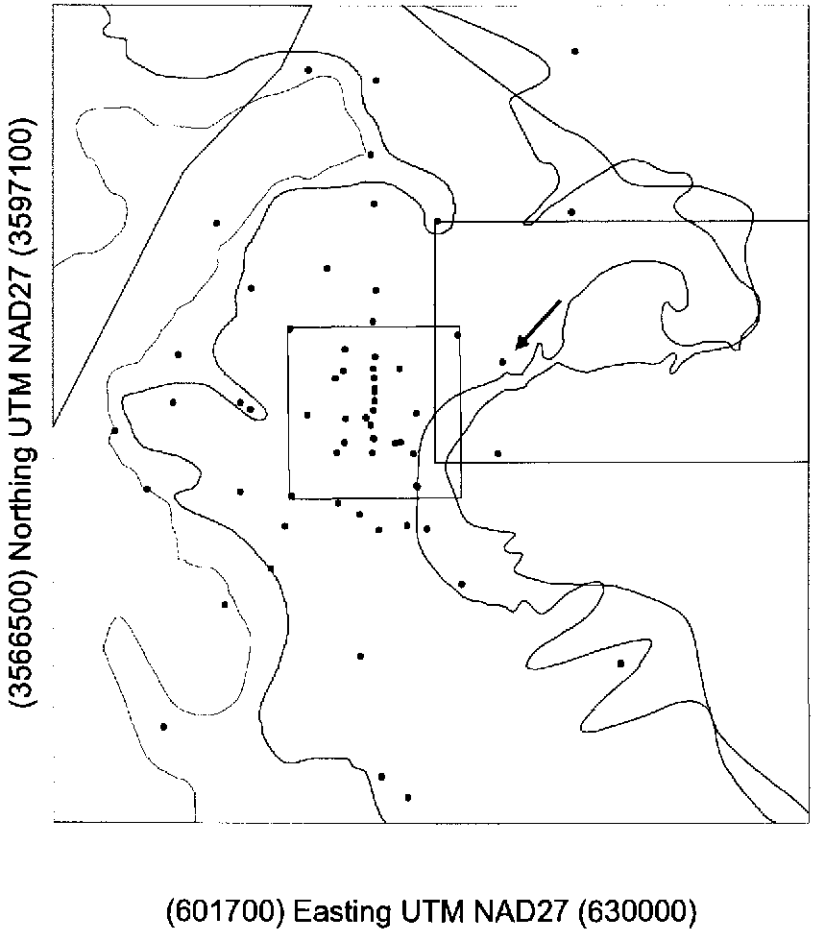


Figure 4-3. Recalibration boundary shown in red to the northeast of the WIPP site. Blue arrow indicates the location of SNL-8. Axis labels are in meters.

4.2.2 Continued Recalibration Activity

After examination of the acceptance criteria (discussed in the next section), some fields were recalibrated again, using the same recalibration process but holding none of the parameter values fixed at previously calculated values. This process essentially added some additional calibration iterations to these fields. This was only done on a few fields that were now well within the acceptance criteria for steady-state heads, and just outside the acceptance criteria for transient tests. The intent of this additional calibration was to increase the quality of the transient fits to get a total of 100 fields that met both the steady-state and transient calibration requirements. This re-re-calibration process was continued until 100 fields were obtained that met the requirements, and it did not always improve fits (i.e., in some cases the fields were already as well calibrated as was possible given the number of pilot points and observations and initial conditions).

4.3 Evaluation of Impact of Multiple Calibration Processes

Because some fields were calibrated only once (set B: 50 fields following correction of the steady-state values), some fields were calibrated once and then underwent a localized recalibration (set A: the 150 first fields calibrated), and some fields even underwent a second round of calibration (set C: 15 fields), it is natural to wonder what impact this may have had on the final selection of fields. The list of fields, and the set they belong to, is presented in Appendix E.

Because the final selection process did not look at which set of fields the results were taken from, the mix of fields should be similar to a random selection if the calibration processes were producing equivalent results. The random selection of fields from set B can be modeled as a binomial distribution with the p-value of 0.25 and n=100. If the results are within the 95% confidence interval for a random selection of fields, then there should be between 17 and 33 fields selected from the set B. The final results used 83 fields from the set A and C, and the remaining 17 were selected from set B. This is within the confidence interval, so it is concluded that the different processes had no impact on the selection of the final fields. The selection of fields from set C versus those from set A can be modeled the same way, with a p-value of 0.10 and n=83. The final selection included 10 from set C which is within the confidence interval of 3 to 13 fields, and again the calibration process did not impact the field selection.

Because this mix of final fields is acceptable and came strictly from the cutoff values, and not from any deliberate attempt to select from one group or another, the analysts believe that all 100 fields meeting the acceptance criteria are equally good and equally probable representations of the Culebra. Furthermore, no one calibration procedure is inherently better than any other, provided that the same acceptance criteria are met, so the use of different procedures may actually improve the representation of uncertainty in the final results.

4.4 Selection of Best-Calibrated Fields

The selection criteria for the "best" calibrated fields consisted of comparing the absolute average error of the modeled steady-state heads to a cutoff value, and comparing the absolute average error of the modeled transient responses to a cutoff value. The steady-state and transient criteria were evaluated separately, and only fields that were less than the cutoff value for both sets of tests were selected as the final fields. The final cutoff values used were the mean value of the errors taken across all 200 fields, and

they are presented in Table 4-1. Using the mean values resulted in a set of 102 fields, so the two fields with the largest sum of the two metrics were discarded. In Figure 4-4, the sum of the steady state average errors was graphed against the sum of the transient pumping tests' average errors, and the selected and unselected fields are shown. The trend line shows graphically how PEST allows tradeoffs while keeping the improvement in errors as balanced as possible.

The final, selected field IDs are presented in

Table 4-2. The fields themselves are stored in the "Results" module of the CVS repository for Task 7. Each field is identified as "*fieldID_Parameter*_field.mod", and they are MODFLOW-ready equivalent files (non-log transformed, conductivity instead of transmissivity, specific storage instead of storativity). The fields are in separate directories, "R_fields", "S_fields", "K_fields" and "A_fields". Also included in the Results module are the flow budget files used as input to DTRKMF and the particle tracks produced by DTRKMF, as is discussed in Section 5.1.

Table 4-1. Cutoff values for final field selection.

Test Type	Average Error Selection Cutoff
Steady State	0.699 m
Transient Pumping Response	0.164 m

Table 4-2. Final selected field identifiers.

r001	r055	r207	r652
r002	r058	r256	r655
r004	r059	r260	r657
r006	r060	r273	r664
r007	r061	r276	r669
r009	r064	r279	r694
r010	r070	r298	r707
r012	r073	r327	r727
r013	r074	r328	r752
r017	r076	r361	r791
r024	r078	r431	r806
r027	r082	r440	r808
r028	r083	r465	r809
r029	r084	r486	r814
r032	r090	r489	r823
r034	r092	r506	r861
r037	r095	r508	r883
r038	r097	r511	r902
r040	r098	r515	r910
r041	r102	r522	r921
r045	r104	r568	r922
r051	r137	r571	r940
r052	r142	r631	r981

r053	r191	r634	r982
r054	r203	r640	r984

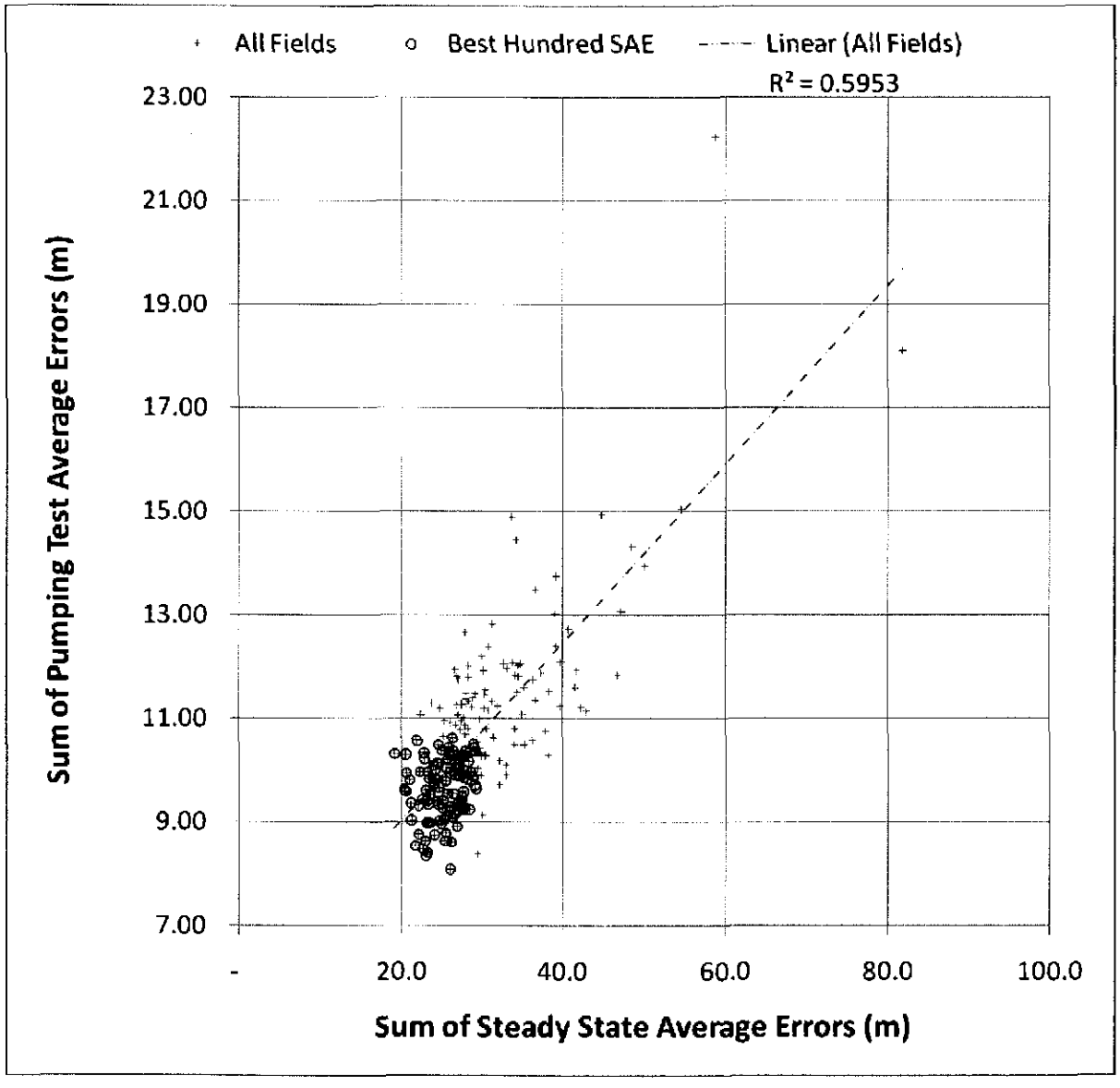


Figure 4-4. Selection of best fields from all fields - graph of steady-state errors vs. pumping test errors.

5 Subtask C – Post Calibration Analysis

Subtask C consisted of calculating travel times, performing analyses on the selected fields, and examining the calibrated forward model outputs. The full results of the forward model outputs are presented in Appendix C (steady-state), Appendix D (pumping tests), and tabular results are presented in Appendix E; this will be reiterated in the appropriate subsections.

5.1 Calculation of Advective Transport Travel Time CDF

The advective travel time for a particle released at C-2737 over the center of the WIPP disposal panels in each field was calculated in the same way as was done in AP-088 Task 4 (McKenna & Hart, 2003). The streamline particle tracking code DTRKMF was used to calculate the path and travel time to reach the WIPP LWB from the release point. These times were then collected and presented as a cumulative distribution function. The only file needed from the calibration results to perform this analysis is the steady-state water budget file, modeled_flow.bud. This file was stored in the Outputs directory for each calibrated field, and was moved to the Results directory following the transport analysis. This particular analysis was also done under run-control, and was executed by the run-master and the results extracted from CVS by the analysts.

The calculation was done with Culebra thickness equal to 7.75 m, and the porosity constant at 0.16 (WIPP parameter DPOROS). The travel times were then scaled to a 4-m-thick Culebra by assuming that all flux through the entire 7.75-m Culebra thickness was instead focused through only the lower 4 m of Culebra. (As discussed in DOE (U.S. Department of Energy), 1996, Appendix MASS, MASS Attachment 15-6, radionuclide transport through the Culebra is modeled as occurring only through the lower 4 m of the Culebra.) The complete list of average observation errors, travel times for 7.75-m thickness, and the calculated travel times for a 4-m thickness are presented in Appendix E. The CDF of the 4-m travel times is presented in Figure 5-1. Figure 5-1 also shows the CDF of travel times from the CCA (DOE, 1996) and CRA-2004 (DOE, 2004). The median travel time for the current fields from the Task 7 analysis is less than 300 years more than the CCA median time, and, like the CCA, 33% of the fields produce travel times greater than 10,000 years.

In addition to the travel time, DTRKMF provides the particle streamlines. In Figure 5-2 and Figure 5-3, the particle tracks for the 100 final selected fields are presented. Figure 5-2 shows the travel up to the WIPP land withdrawal boundary, with the grayscale color representing the time for the particle to reach that point on the streamline. Figure 5-3 shows the number of fields producing streamlines that pass through a particular cell. In addition, Figure 5-3 shows the streamlines continuing to the model domain boundary.

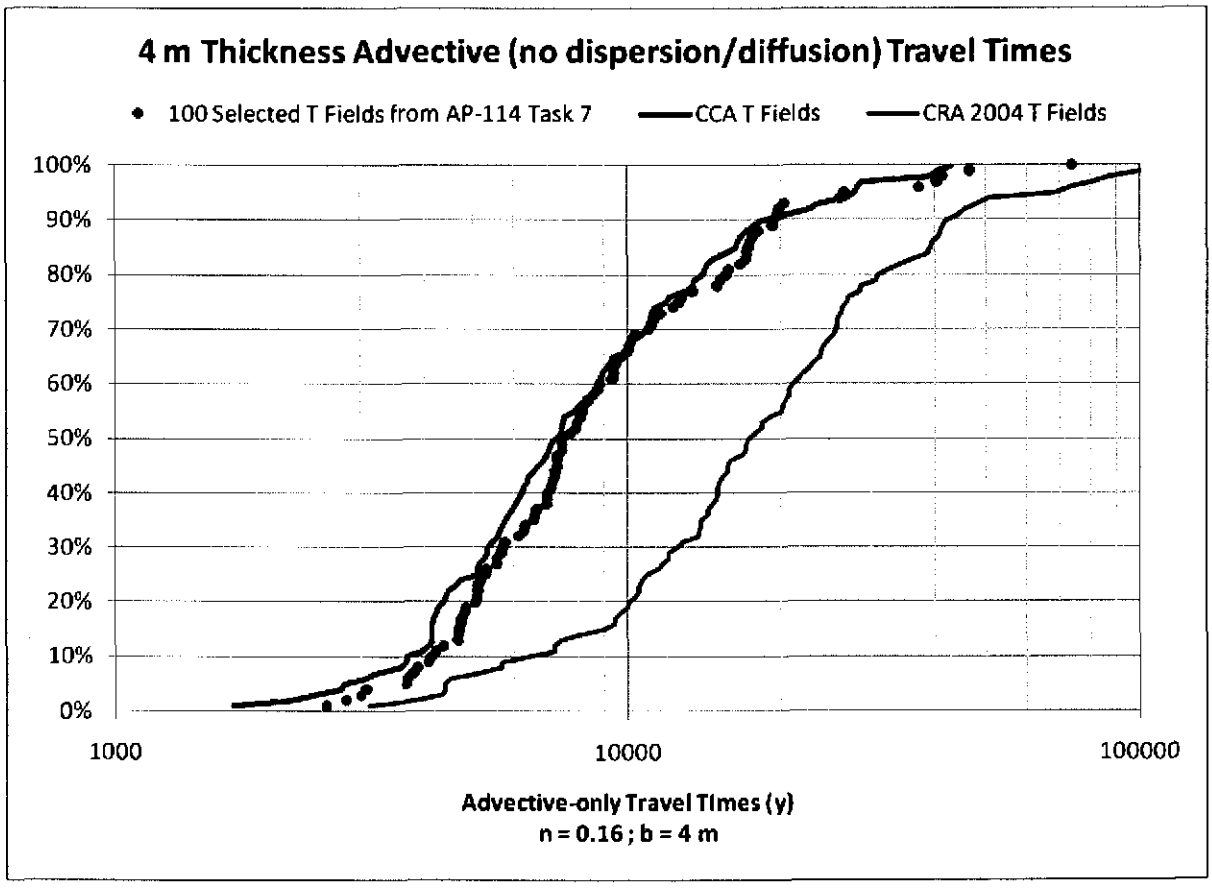


Figure 5-1. Advective travel times to reach WIPP LWB.

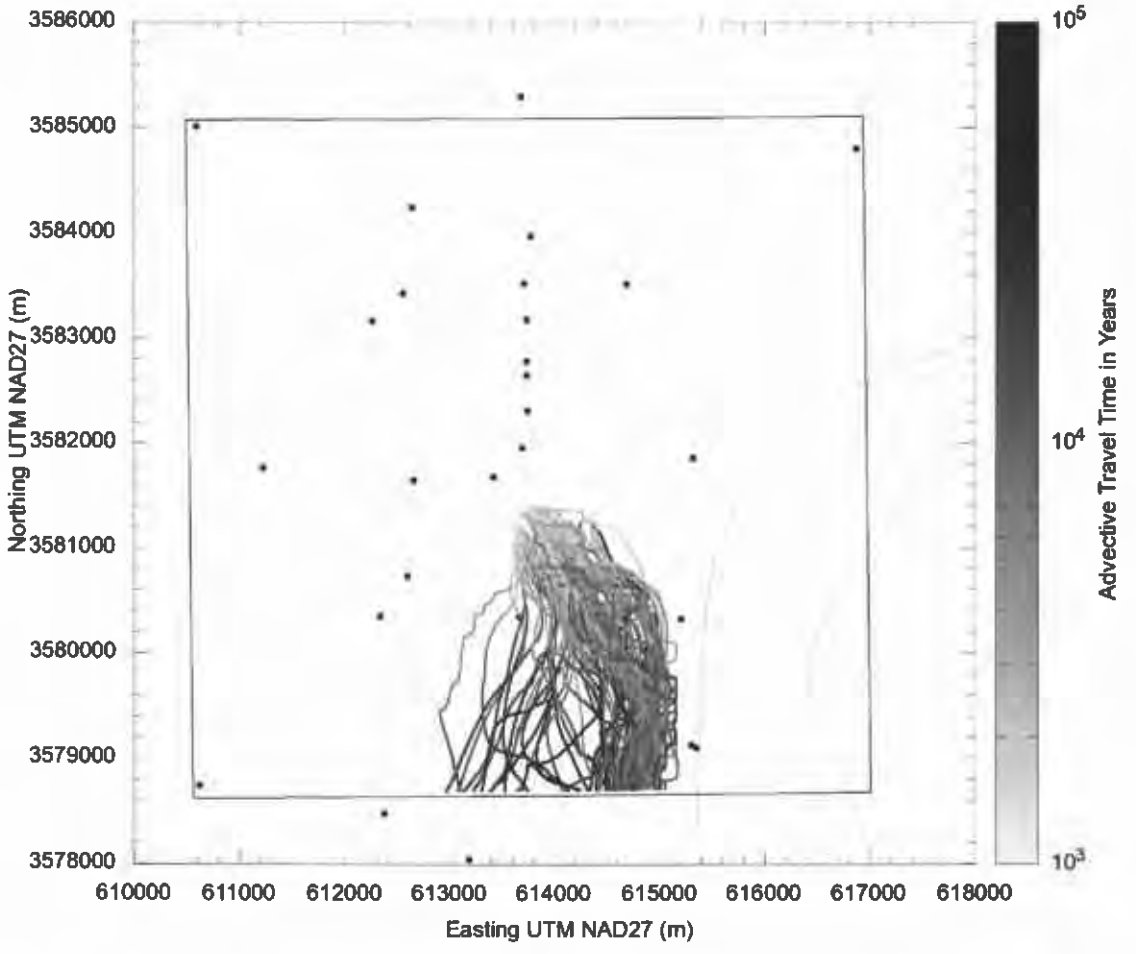


Figure 5-2. Particle streamlines to the LWB for the 100 selected fields. The effects of the high-T channel can be seen in the flow paths.

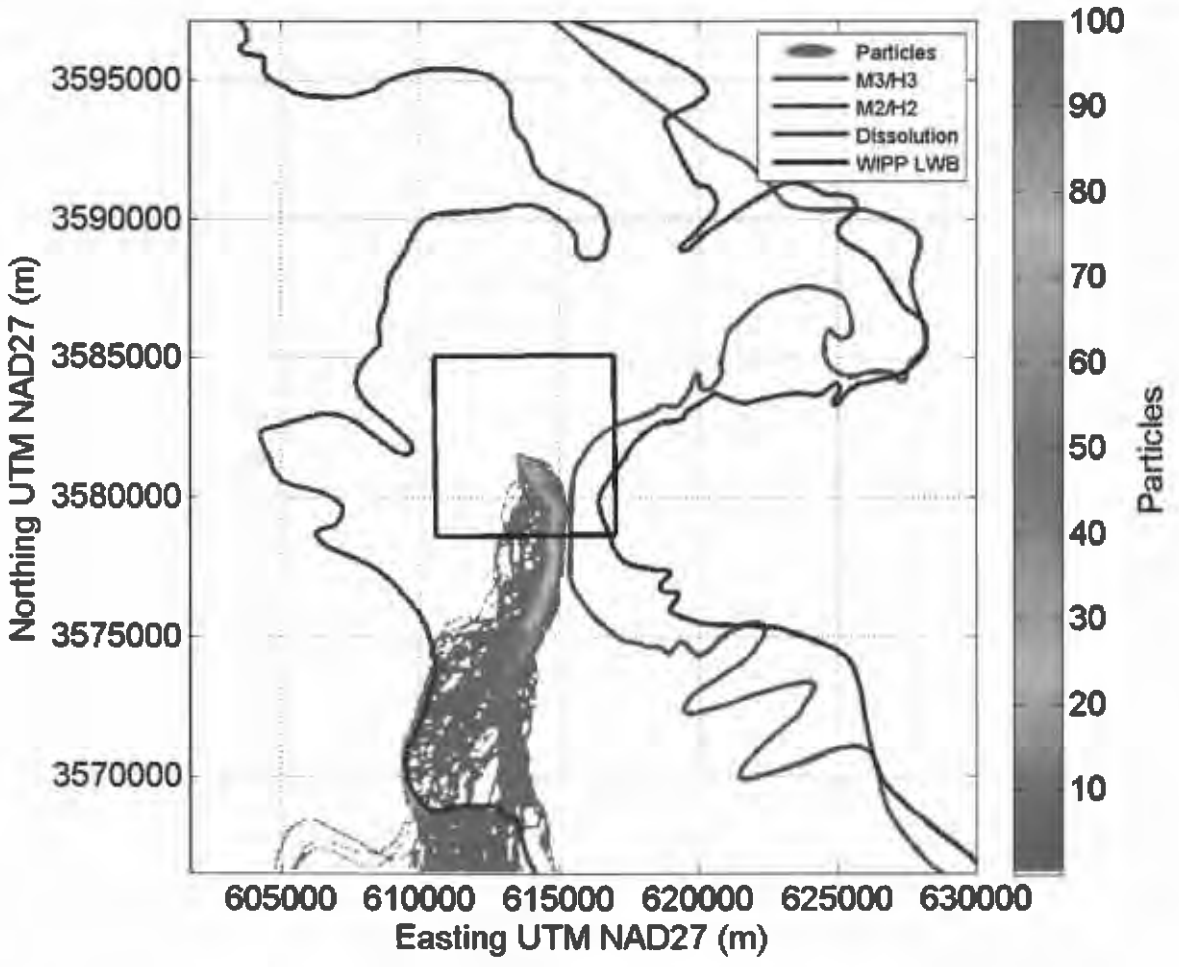


Figure 5-3. Particle density in each cell for the 100 final selected fields.

5.2 Statistical Analysis of Resulting T, A, and S Fields

There are many analyses that can be done on the resulting fields from the calibrations. However, the most relevant are the mean and standard deviations of the final 100 fields, the bulk transmissivity of the final fields compared to the bulk transmissivity of the starting base fields, and the advective travel times presented previously. This section will present the average fields and the bulk transmissivity comparisons.

5.2.1 Final Transmissivity and Anisotropy Fields

While the anisotropy will be presented separately, the transmissivity values presented in this section are the *effective* transmissivity values, which include the anisotropy. (Effective transmissivity ($\log_{10} T_e$) was calculated as $\log_{10} T_{EW} + \frac{1}{2} \log_{10} A$ – which is the average of $\log_{10}$T in the North-South and East-West directions. See the equation in Section 3.1.2.) The bulk transmissivity, which is the average $\log_{10}$ transmissivity value of all cells in a given zone or zones, was calculated for the central and Salado dissolution zones (zones 0-2) and compared to the bulk transmissivity of the same zones from the base fields. The eastern, very low transmissivity zone (zone 3) was compared separately. The bulk transmissivity values are shown in Table 5-1. The mean effective transmissivity and the standard deviation of transmissivity are presented in map form in Figure 5-4 and Figure 5-5. The mean effective transmissivity map does not show the very low T zone in order to keep the color map sufficiently distinct for the area around the WIPP site.

Because pilot point parameter values were essentially unconstrained for T, some areas in zones 0 and 1 could effectively "change zones" by moving the values in a low-T zone into the range generally considered "high-T" and vice versa. The defining value for "high-T" was set in Hart, Holt & McKenna (2008) to be the bulk transmissivity value of the base fields: -5.41 $\log_{10}$ m^2/s. At each cell, the number of fields whose initial and final T values were in the "high-T" zone was calculated, and the maps of those numbers for the base and calibrated fields are presented in Figure 5-6 and Figure 5-7, respectively. The total number of fields where transmissivity "changed zones" is represented graphically in Figure 5-8 and Figure 5-9. In Figure 5-8 and Figure 5-9, the white regions define the areas where the original T values were "low" or "high," respectively, and could not or did not make the specified change. The two measures shown in these sets of maps provide an indication of how the geologically based conceptual model used to create the base fields was altered by the steady-state and transient hydraulic information.

The mean and standard deviation of anisotropy is presented in Figure 5-10 and Figure 5-11. These maps show that changes in the hydraulic anisotropy can stay constrained between 0.5 and 5.0, and that the most significant and consistent changes occur within the high-T area to the west of the dissolution boundary. In the halite-cemented zone (Zone 3), the head is fixed, and there are no pilot points; an artifact of the parameter field conversion is that the means and standard deviations are not zero for anisotropy in this zone.

Table 5-1. Bulk $\log_{10}$ transmissivity values comparison.

Base field bulk $\log_{10}$ transmissivity (Zones 0-2)	$-5.41\ \log_{10}(m^2/s)$
Calibrated field bulk $\log_{10}$ transmissivity (Zones 0-2)	$-5.02\ \log_{10}(m^2/s)$
Base field bulk $\log_{10}$ transmissivity (Zone 3)	$-11.74\ \log_{10}(m^2/s)$
Calibrated field bulk $\log_{10}$ transmissivity (Zone 3)	$-10.47\ \log_{10}(m^2/s)$

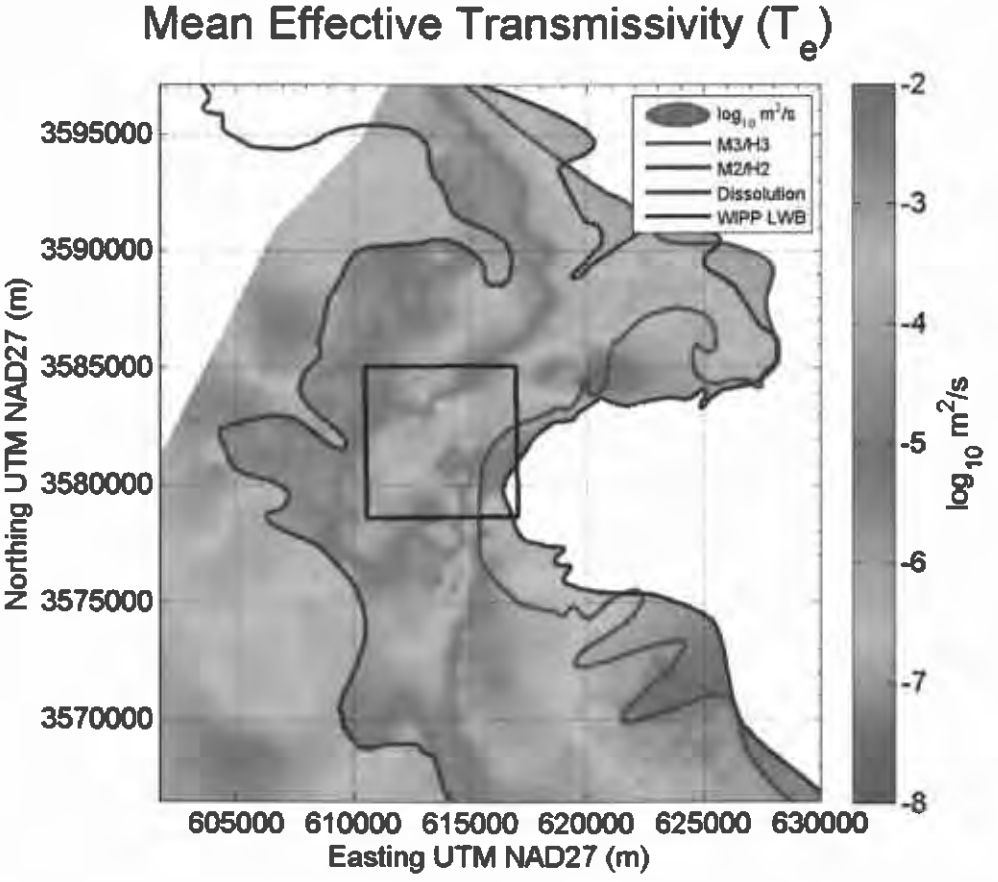


Figure 5-4. Mean effective transmissivity for zones 0-2 across the 100 final selected fields.

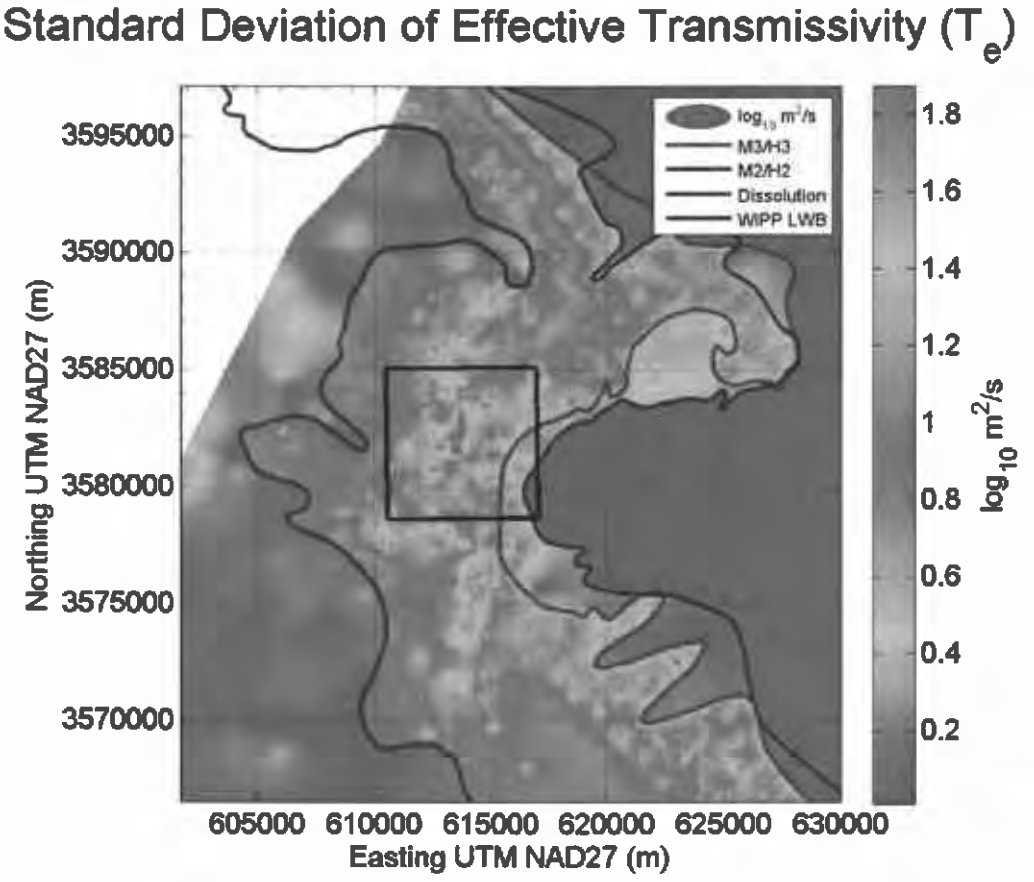


Figure 5-5. Standard deviation of effective transmissivity for all zones across the 100 final selected fields.

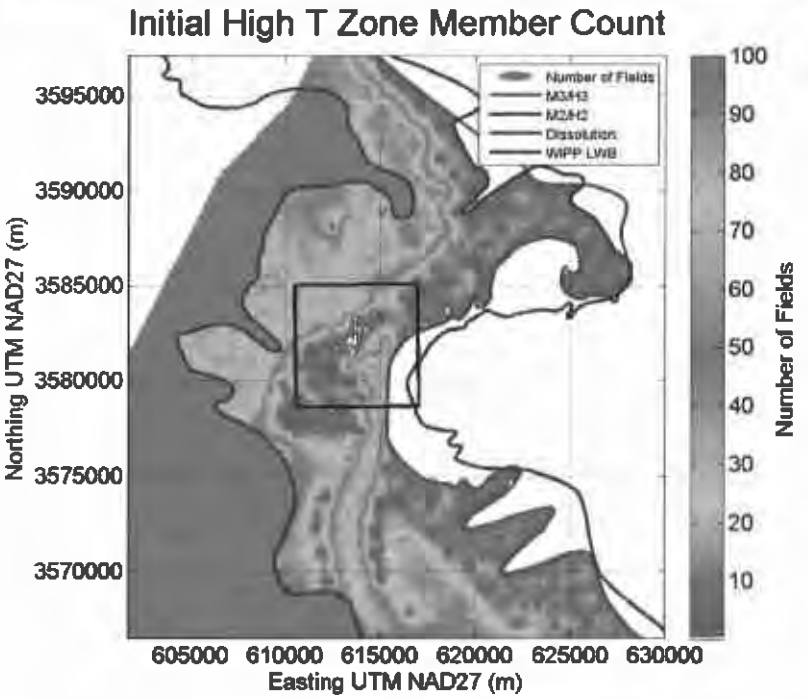


Figure 5-6. Number of fields where a given cell was a member of the "High-T" zone, calculated for the base T values.

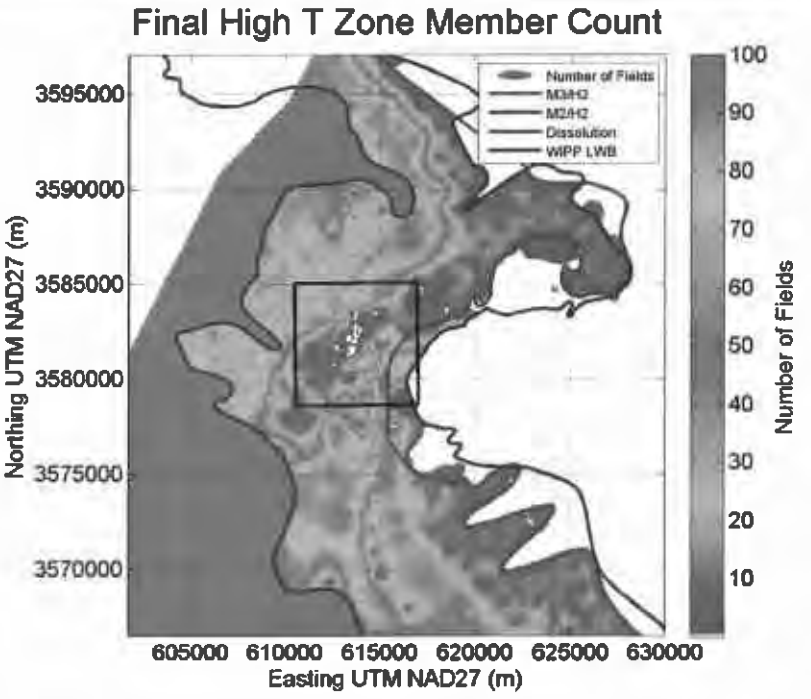


Figure 5-7. Number of fields where a given cell was a member of the "High-T" zone, calculated for the final transmissivity values.

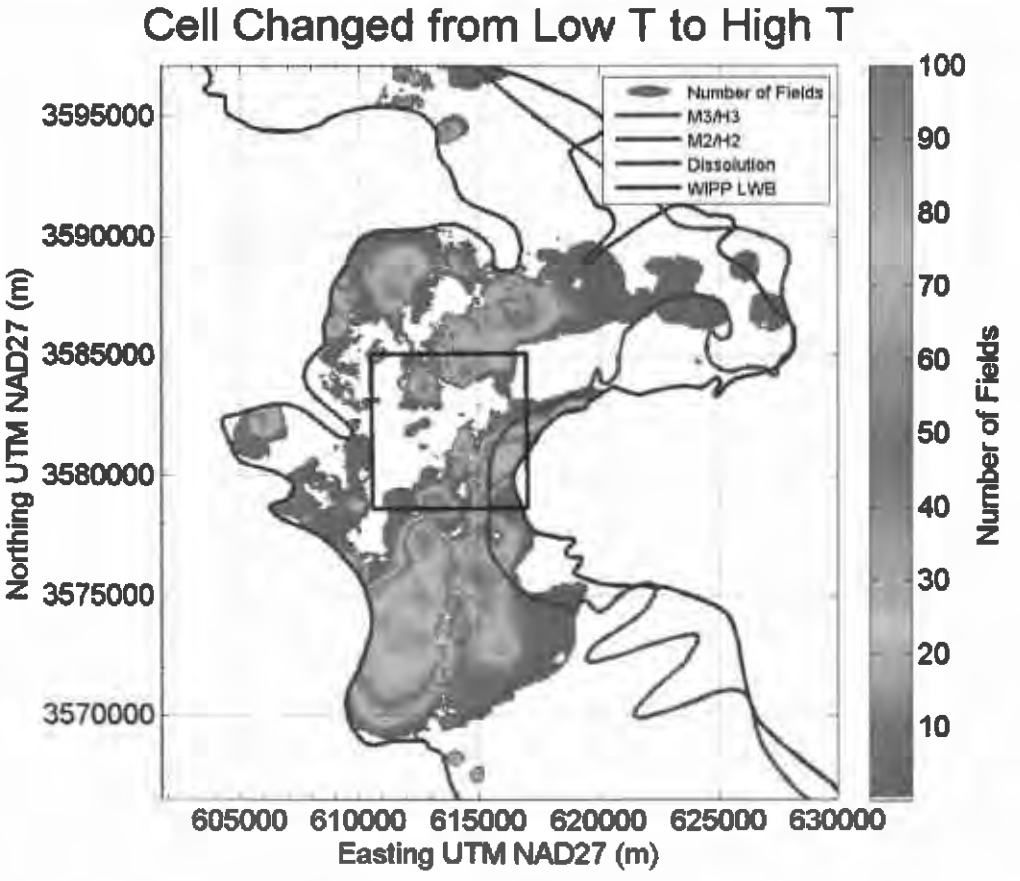


Figure 5-8. Number of fields where cells that started as low-T became high-T. Calculated for each cell.

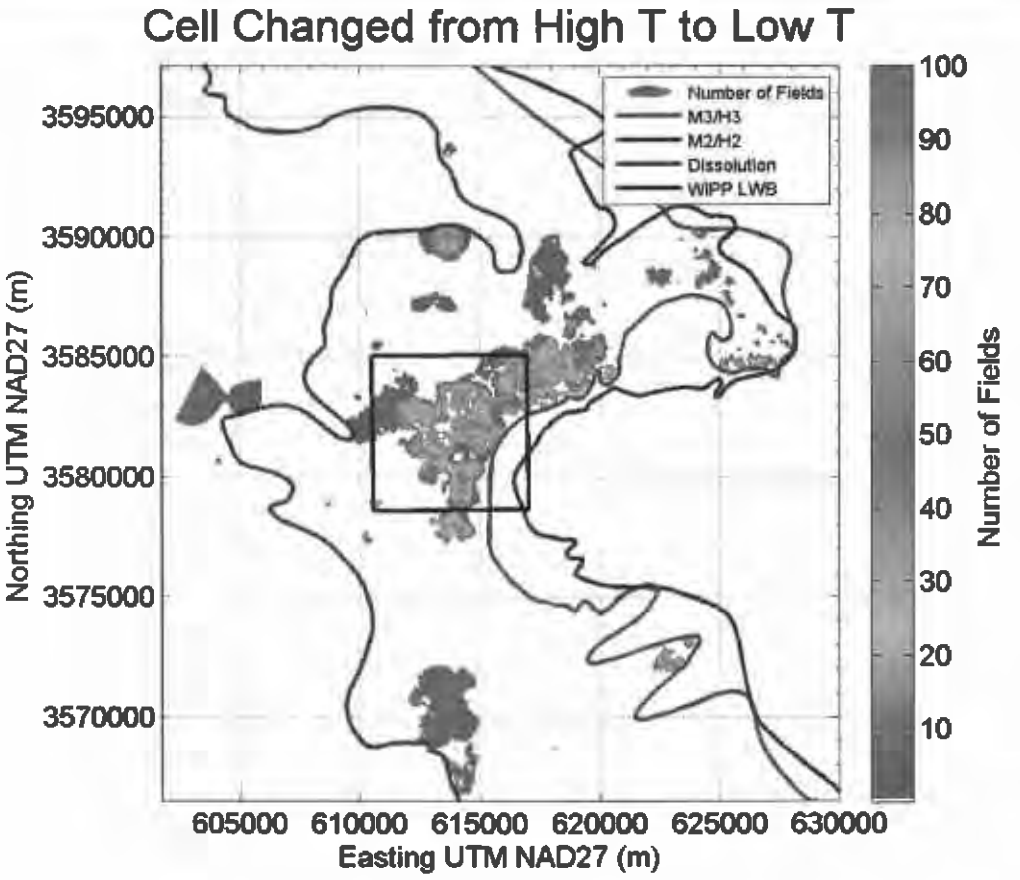


Figure 5-9. Number of fields where cells that started as high T became low T. Calculated for each cell.

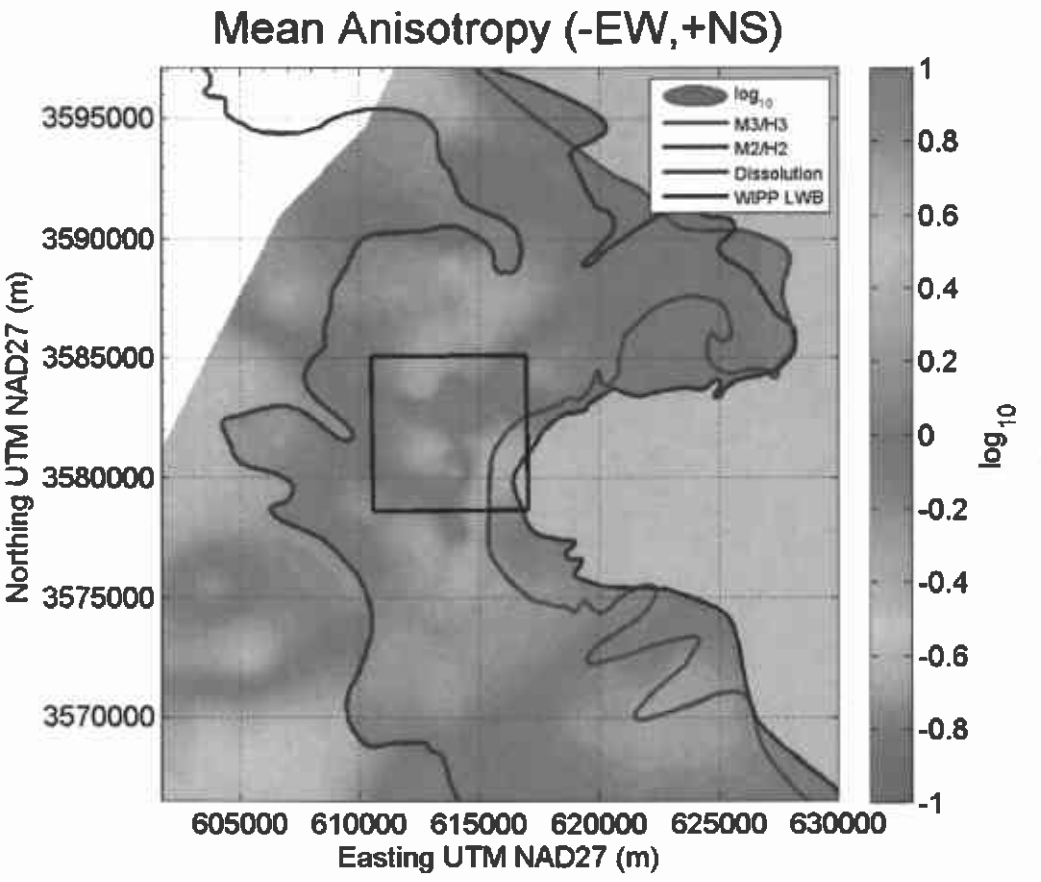


Figure 5-10. Average anisotropy values across the 100 final fields. Positive numbers indicate strong north-south anisotropy, while negative numbers indicate strong east-west anisotropy. Values in zone 3 (halite-bounded) are non-zero because a shifting term was applied for that zone.

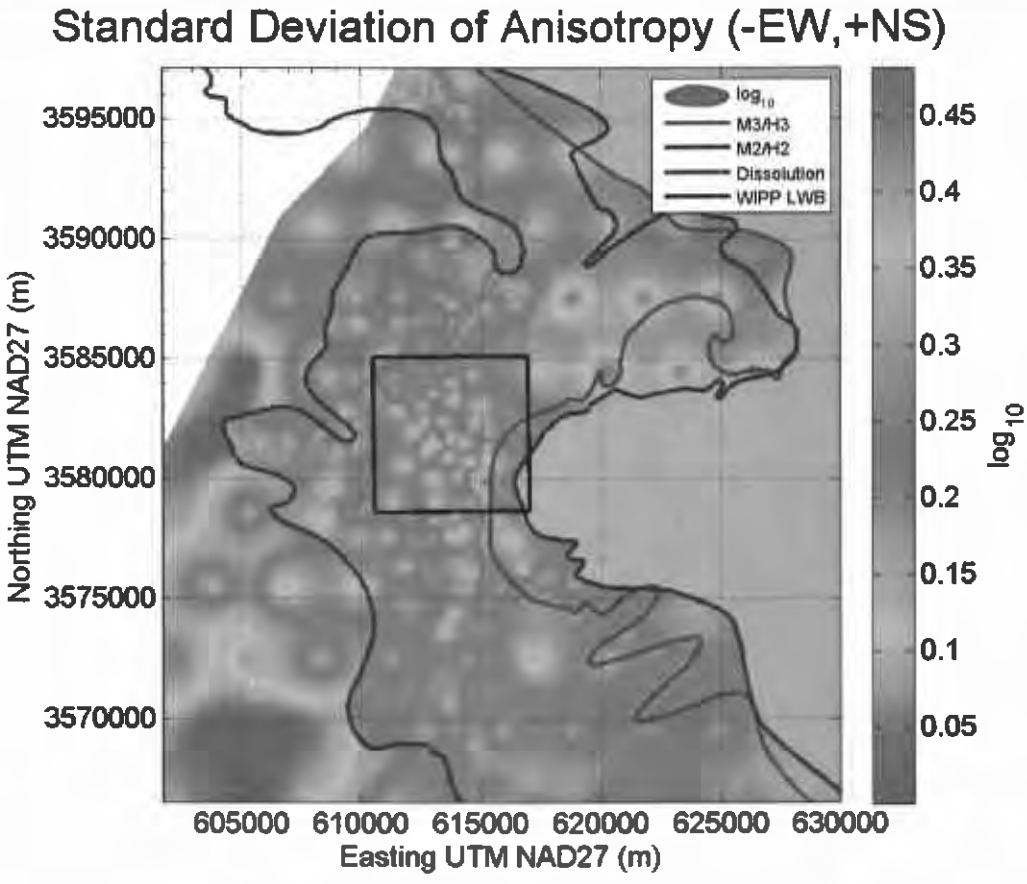


Figure 5-11. Standard deviation of anisotropy values across the 100 final calibrated fields. Values in zone 3 (halite-bounded) are non-zero because a shifting term was applied for that zone.

5.2.2 Final Storativity Values

The mean and standard deviation of the final storativity values are presented in Figure 5-12 and Figure 5-13. The mean storativity values indicate that the overall storativity values in the confined and transitional zones did not change much from the initial values, however the area northwest of P-14 shows high variability in the storativity values used in individual solutions (Figure 5-13). This may have some relation to the relatively poorer fits of the transient data for the WIPP-25 response to the P-14 pumping test in the model.

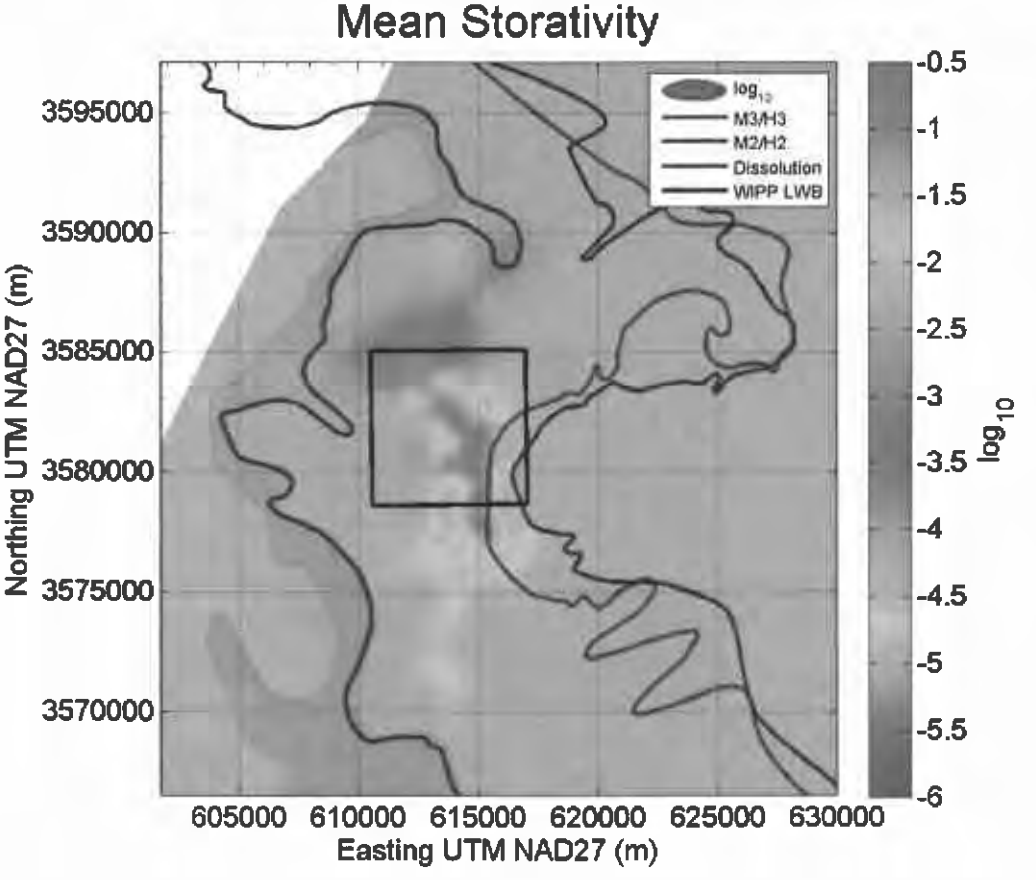


Figure 5-12. Mean storativity values across the 100 final calibrated fields. Values in zone 3 (halite-bounded) are non-zero because a shifting term was applied for that zone.

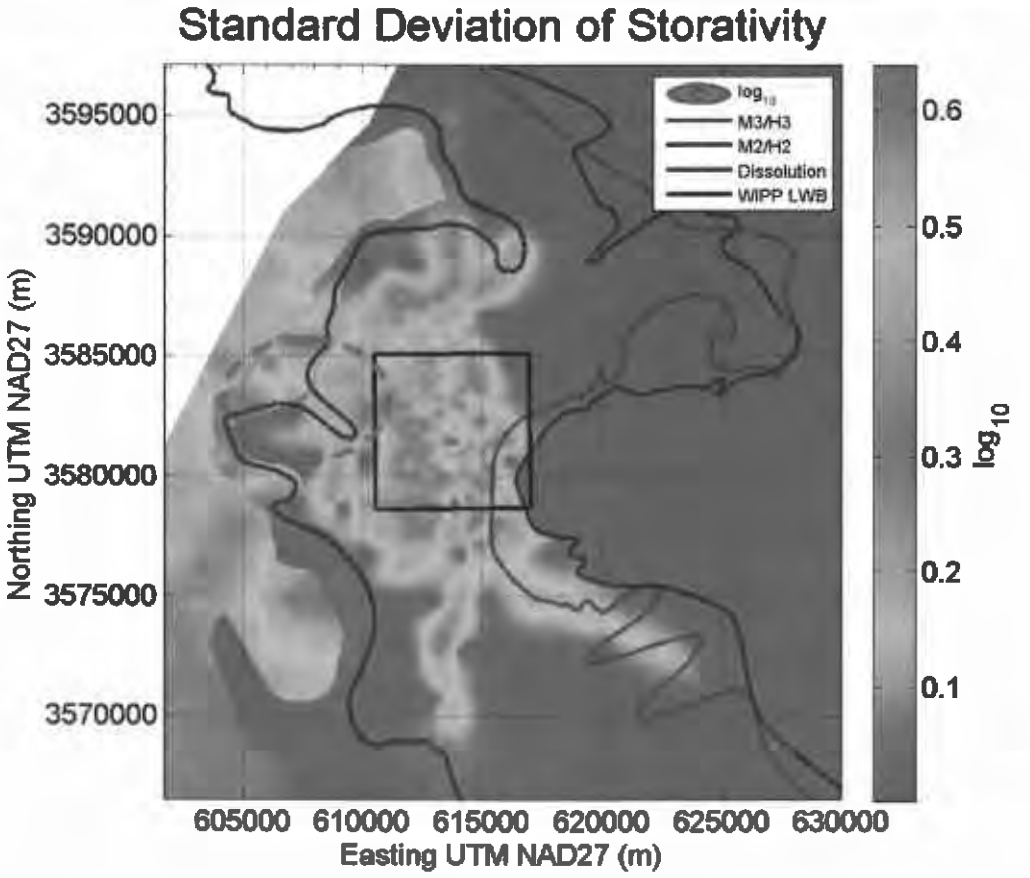


Figure 5-13. Standard deviation of storativity values across the 100 final calibrated fields. Red ellipse shows P-14 to WIPP-25 area of influence. Values in zone 3 (halite-bounded) are non-zero because a shifting term was applied for that zone.

5.2.3 Final Recharge Values

The final recharge values were all less than the initial values of 10^{-11} m/s (3.2×10^{-4} m/yr). Compared to the other parameters, there was very little change in recharge. Because the recharge zone was linear, in addition to the cell-by-cell mapping, a view of the average, minimum and maximum recharge values, in meters per year, is shown as a cross section in the X-direction (across a row) as if looking from the South to the North through the domain in Figure 5-14. Figure 5-15 shows the map of recharge, where all the white cells represent no recharge.

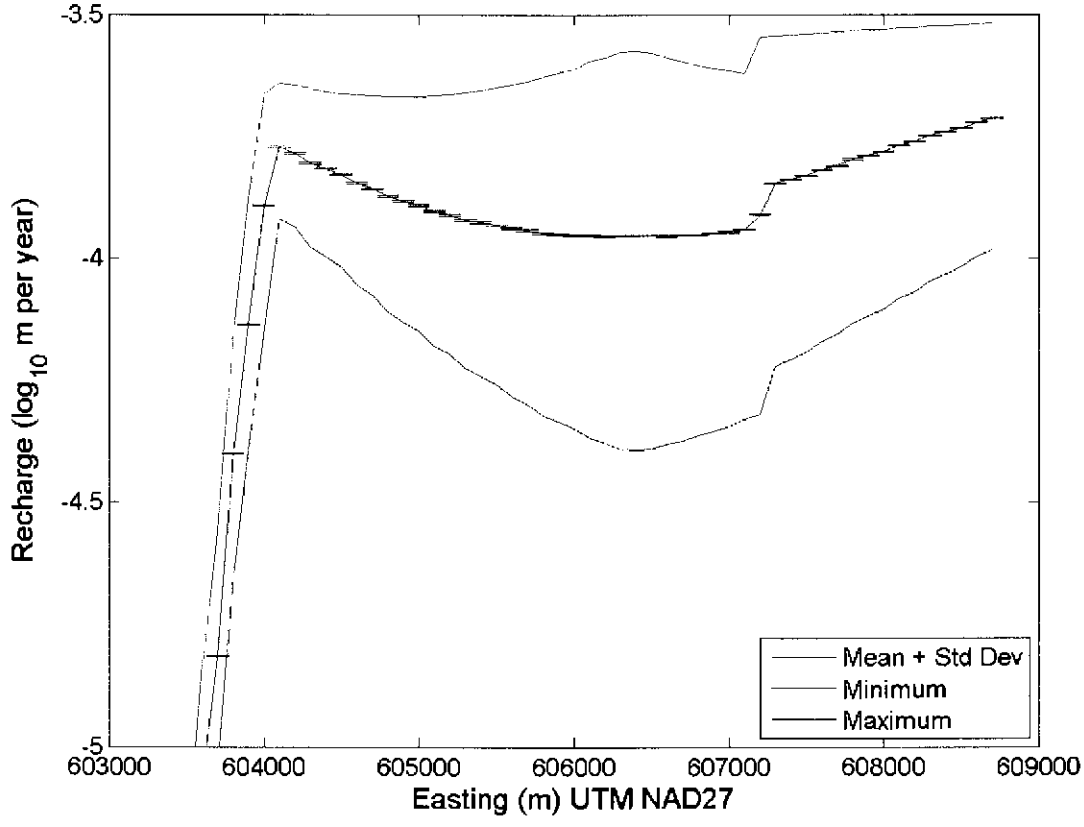


Figure 5-14. Recharge as viewed through columns from the south. The initial value was set at -3.5 $\log_{10}$ m/year, which is the maximum y-value on the graph. The sharp dropoff to the west is the transition to the single fixed-recharge point of -11.5 $\log_{10}$ m/year (interpreted as 0 by REAL2MOD).

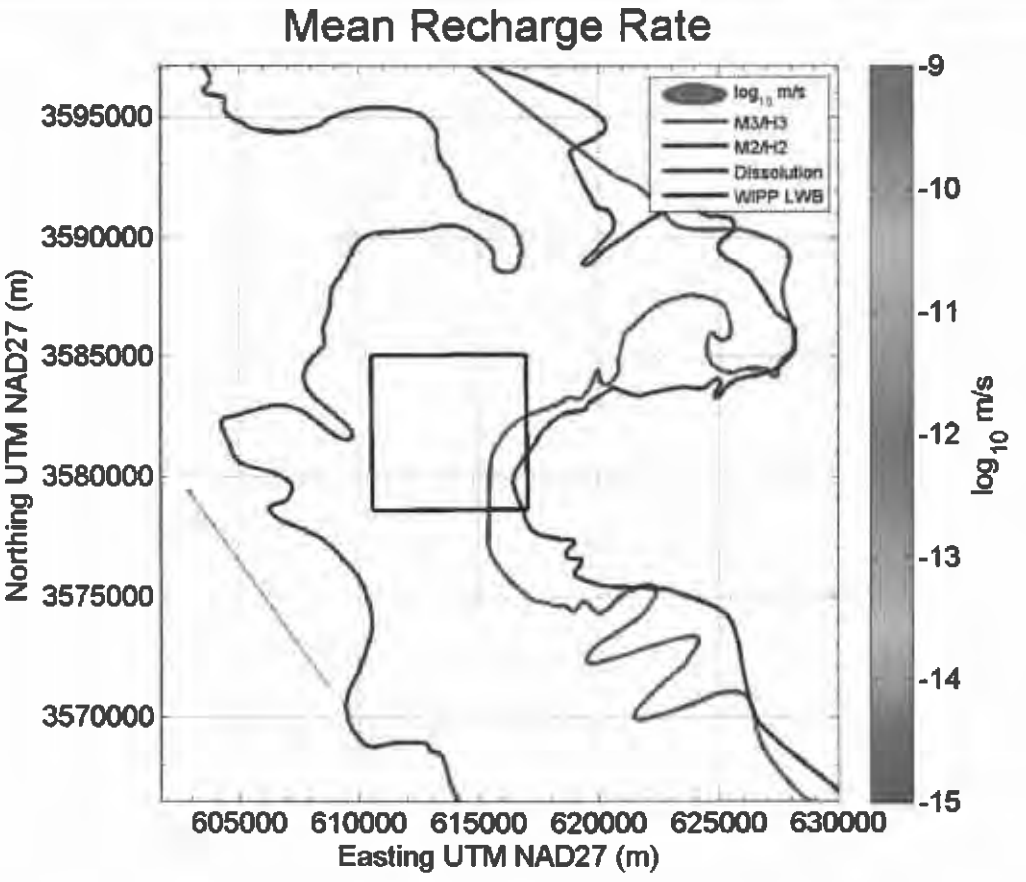


Figure 5-15. Map of average recharge across final 100 selected fields.

5.3 Forward Model Results using the Calibrated Fields

The two main divisions of the results are the steady-state head results and the pumping test results. The results presented here only represent the 100 final selected fields, and therefore the maximum error is limited by the selection criteria described in Section 4.4: an average steady-state error of less than 0.700 m and an average pumping test observation error of less than 0.164 m.

Figure 5-16 shows the modeled steady-state head values plotted against the measured head values. The one-to-one correspondence line shows the ideal match, and the modeled results are presented as box-and-whisker plots at each observation well. Figure 5-17 shows all the head errors for all 100 fields as a histogram of error values for steady-state head. Additional figures and tables are presented in Appendix C. The estimated measurement error can be modeled as a zero-mean Gaussian distribution with a standard deviation of 0.10 m (McKenna & Wahi, 2006). The measurement error distribution curve is included in Figure 5-17.

Graphs for each of the transient pumping test results are presented in Appendix D. The average error of the final fields ranged from 0.12 m to 0.164 m across all tests, with an average error of 0.15 m. The maximum error for a single observation well ranged from 0.005 m to 2.5 m, with an average of 0.36 m as the maximum error at a given observation well.

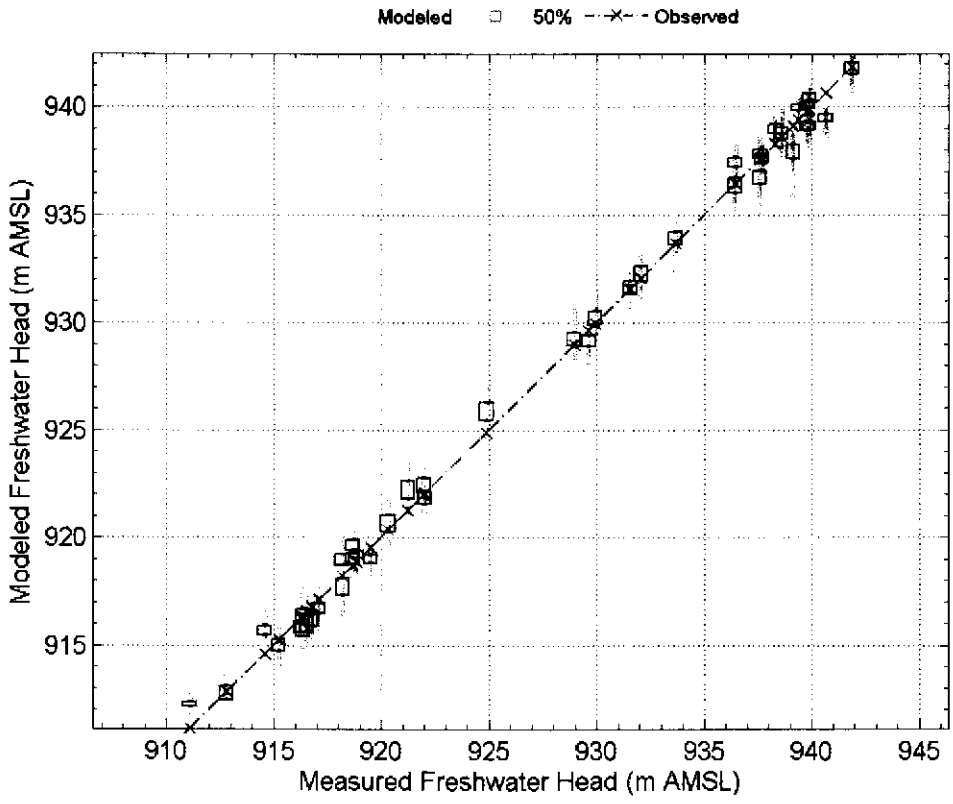


Figure 5-16. Results for 42 of the 44 total steady-state head measurements for the 100 selected fields. Observed field values are indicated by cross hatches along the 1-to-1 line. Wells SNL-6 and SNL-15 are located in the fixed-head region of the model, and they are not shown on this graph.

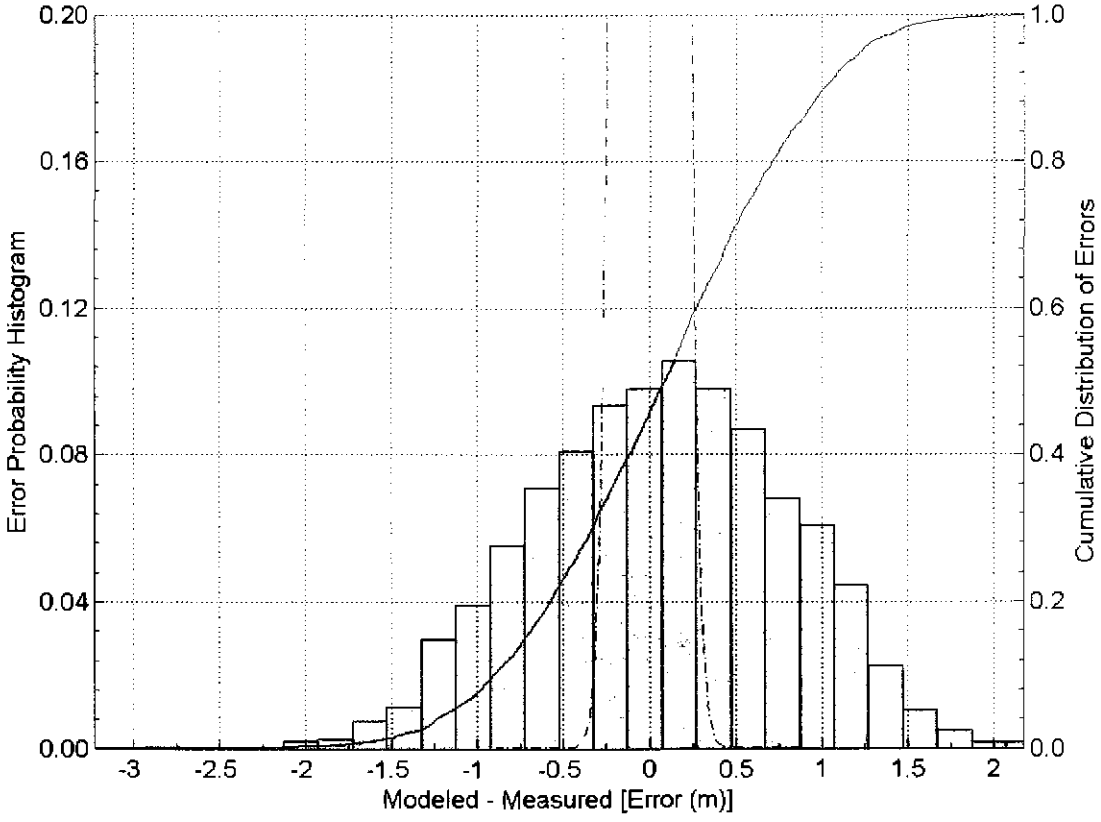


Figure 5-17. Histogram of steady-state head errors for the 100 selected fields. Wells SNL-6 and SNL-15 are not included. Red dashed line is the +/- 3σ section of the measurement error PDF. Seeming slight skew to right is an artifact of the binning.

6 Conclusion

The calibration of the Culebra T fields accomplished in Task 7 of AP-114 was completed by integrating a comprehensive geologic conceptual model (developed in Hart, Holt & McKenna, 2008) with a large set of hydraulic data, including nine pumping tests with a total of 65 observation wells and 1300+ measurements, along with 44 steady-state head measurements. The large number of parameter values and pilot points used was only possible due to the large computing resources available and the advances in inverse calibration techniques that were included in the software used. In general, calibration reduced the model errors by a factor of 5 to 7 relative to the uncalibrated base fields (as is evidenced in the run log files).

7 References

Beauheim, RL 1987a, "Analysis of Pumping Tests of the Culebra Dolomite Conducted at the H-3 Hydropad at the Waste Isolation Pilot Plant (WIPP) Site", SAND86-2311, Sandia National Laboratories.

Beauheim, RL 1987b, "Interpretation of the WIPP-13 Multipad Pumping Test of the Culebra Dolomite at the Waste Isolation Pilot Plant (WIPP) Site", SAND87-2456, Sandia National Laboratories.

Beauheim, RL 1989, "Interpretation of H-11b4 Hydraulic Tests and the H-11 Multipad Pumping Test of the Culebra Dolomite at the Waste Isolation Pilot Plant (WIPP) Site", SAND89-0536, Sandia National Laboratories.

Beauheim, RL 2008, "Analysis Plan for Evaluation and Recalibration of Culebra Transmissivity Fields, AP-114, Revision 1", ERMS# 548162, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Beauheim, RL 2009, "Changes to Culebra T-Field Calibration Procedures under AP-114 Task 7", ERMS# 551437, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Beauheim, RL & Ruskauff, GJ 1998, "Analysis of Hydraulic Tests of the Culebra and Magenta Dolomites and Dewey Lake Redbeds Conducted at the Waste Isolation Pilot Plant Site", SAND98-0049, Sandia National Laboratories.

Bowman, DO & Roberts, RM 2009, "Analysis Report for AP-070, Analysis of Culebra and Magenta Hydraulic Tests Performed Between January 2005 and August 2008", ERMS# 550906, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Chavez, M 2006, "Procedure NP 19-1, Revision 12, SOFTWARE REQUIREMENTS", ERMS# 543743, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Chavez, M 2008, "Procedure NP 9-1, Revision 7, ANALYSES", ERMS# 549225, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

DOE (U.S. Department of Energy) 1996, "Title 40 CFR Part 191 Compliance Certification Application for the Waste Isolation Pilot Plant", DOE/CAO-1996-2184, U.S. DOE, Carlsbad Area Office, Carlsbad, NM.

DOE (U.S. Department of Energy) 2004, "Title 40 CFR Part 191 Subparts B and C Compliance Recertification Application for the Waste Isolation Pilot Plant, March 2004", DOE/WIPP 2004-3231, U.S. DOE Waste Isolation Pilot Plant, Carlsbad Field Office, Carlsbad, NM.

Doherty, J 2004, "PEST: Model Independent Parameter Estimation. User's Manual: 5th Edition", Watermark Numerical Computing.

Harbaugh, AW, Banta, ER, Hill, MC & McDonald, MG 2000, "MODFLOW-2000, the U.S. Geological Survey Modular Ground-water Model User Guide to Modularization Concepts and the Gound-water Flow Process", Open-File Report 00-92, U.S. Geological Survey.

Hart, DB, Holt, RM & McKenna, SA 2008, "Analysis Report for Task 5 of AP-114: Generation of Revised Base Transmissivity Fields", ERMS# 541153, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Holt, RM 1997, "Conceptual Model for Transport Processes in the Culebra Dolomite Member, Rustler Formation", SAND97-0194, Sandia National Laboratories, Albuquerque, NM.

Johnson, PB 2009, "Routine Calculations Report In Support of Task 6 of AP-114, Potentiometric Surface, Adjusted to Equivalent Freshwater Heads, of the Culebra Dolomite Member of the Rustler Formation near the WIPP Site, May 2007, Revision 2", ERMS# 551116, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Kirchner, T 2008, "User's Guide to CVS, ReadScript.py, and Related Utilities, Version 1.00", ERMS# 550579, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

McKenna, SA & Hart, DB 2003, "Analysis Report for Task 4 of AP-088: Conditioning of Base T Fields to Transient Heads", ERMS# 531124, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

McKenna, SA & Wahi, A 2006, "Local Hydraulic Gradient Estimator Analysis of Long-Term Monitoring Networks", *Graund Water*, vol 44, no. 5, pp. 723-731.

Mehl, SW 2001, "MODFLOW-2000, The U.S. Geological Survey Modular Ground-water Model User Guide To The Link-AMG (LMG) Package For Solving Matrix Equations Using An Algebraic Multigrid Solver", Open-File Report 01-177, U.S. Geological Survey.

Powers, DW 2006, "Analysis Report, Task 1B of AP-114, Identify Possible Area of Recharge to the Culebra West and South of WIPP", ERMS# 543094, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Roberts, RM 2006, "Analysis Report for AP-070, Analysis of Culebra Pumping Tests Performed Between December 2003 and August 2005", ERMS# 543901, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Roberts, RM 2007, "Analysis Report for AP-070, Analysis of Culebra Hydraulic Tests Performed Between June 2006 and September 2007", ERMS# 547418, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Toll, NJ & Johnson, PB 2006a, "Routine Calculations Report In Support of Task 6 of AP-114, SNL-14 August 2005 Pumping Test Observation Well Data Processing, Summary of Files", ERMS# 543371, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

Toll, NJ & Johnson, PB 2006b, "Routine Calculations Report In Support of Task 6 of AP-114, WIPP-11 February 2005 Pumping Test Observation Well Data Processing - Summary of Files", ERMS# 543651, Sandia National Laboratories WIPP Records Center, Carlsbad, NM.

[THIS PAGE INTENTIONALLY LEFT BLANK]

Appendices

The shorthand notation "CVS://" will be used to represent ":ext:alice.sandia.gov:/nfs/data/CVSLIB" in text and graphics. The first directory listed after the prefix is the CVS repository, and "::" will be used between the repository name and the first module directory. For example, "CVS://Tfields::Inputs/data" refers to the "data" subdirectory of the "Inputs" module in the "Tfields" repository located on alice.sandia.gov.

The notation "r???" is used to represent the realization identifier for a particular base field. This identifier is used throughout the calibration process to indicate both the base field and resulting data files.

Appendix A: Routine Calculation – Creation of Initial Head Field

The creation of the initial head field was accomplished using MATLAB® 7.7.0 (r2008b). The two input files used were "elev_surface.mod" and "init_bnds.inf" from the "Inputs/data" module in CVS. The output file was "init_head.mod" and is also contained in the "Inputs/data" module. The MATLAB script that was used is presented in Listing A-1, and is available at CVS://Tfields::Analysis/MATALAB/create_heads.m, and the output, as captured using the "Publish" feature in MATLAB, is presented following the table in Figure A-1 to Figure A-6, as noted in the script comments. The equations used in this script are presented in Section 3.4.

Listing A-1 - Script used to create the initial head surface for the steady-state model.

```
%% Configure the grid, based on a zero-center with I,J indices. See Figure A-1
[X,Y] = meshgrid(-142:141,-153:153);
Z = 928 + 8 .* ( (Y./153) + sign(Y./153).*(abs(Y./153).^(1/2)) )  + ...
  1.2 .* ( - (X./142).^2 + (X./142).^3 - (X./142)  );
mesh (X,Y,Z);

%% Load the surface values for head in the eastern, fixed head region. Figure A-2
load elev_surface.mod;
head = reshape(elev_surface,284,307)';
head = head(end:-1:1,:);
mesh(X,Y,head);

%% Load the flow boundaries. Figure A-3
load init_bnds.inf;
bnds = init_bnds(end:-1:1,:);
mesh(X,Y,bnds);

%% Clean up the flow boundaries. Figure A-4
bnds(bnds==1)=0;
bnds(bnds==-1)=1;
mesh(X,Y,bnds);

%% Create the new head values. Figure A-5
newHead = head.*bnds + Z.*~bnds;
mesh(X,Y,newHead);

%% Fix the edges. Figure A-6
newHead(:,1) = Z(:,1);
newHead(1,:) = Z(1,:);
newHead(307,1:141) = Z(307,1:141);
newHead(1:4,end) = Z(1:4,end);
mesh(X,Y,newHead);

%% Save final head field
newHead = newHead(end:-1:1,:);
save('init_head.mod','newHead','-ASCII')
```

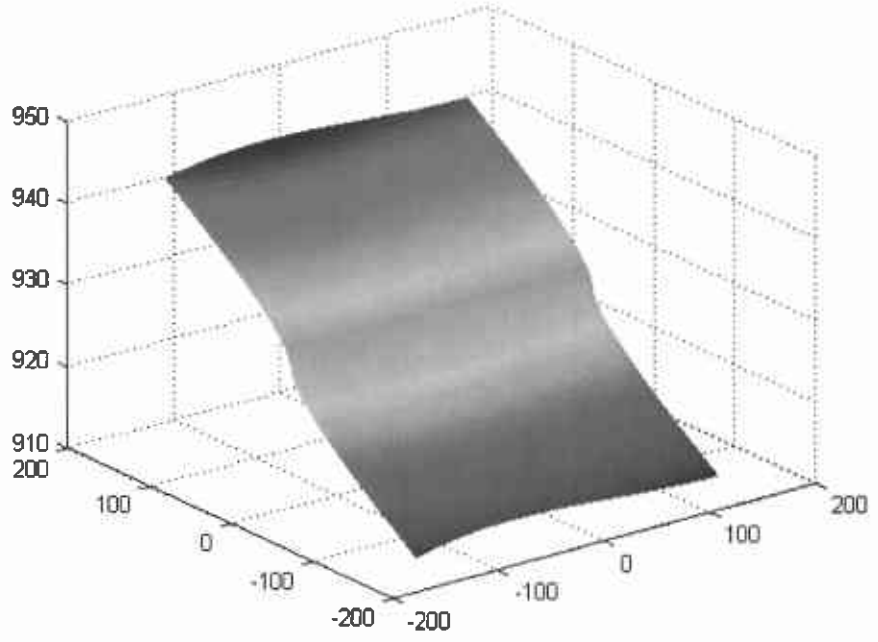


Figure A-1: Initial parametric surface calculated using X and Y.

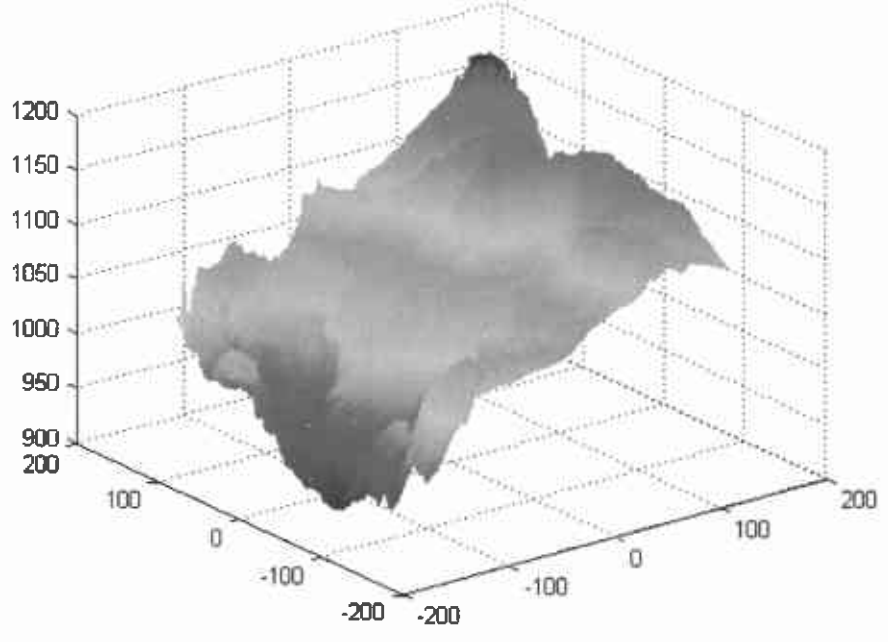


Figure A-2: The ground surface elevation that was used for the low-T zone head.

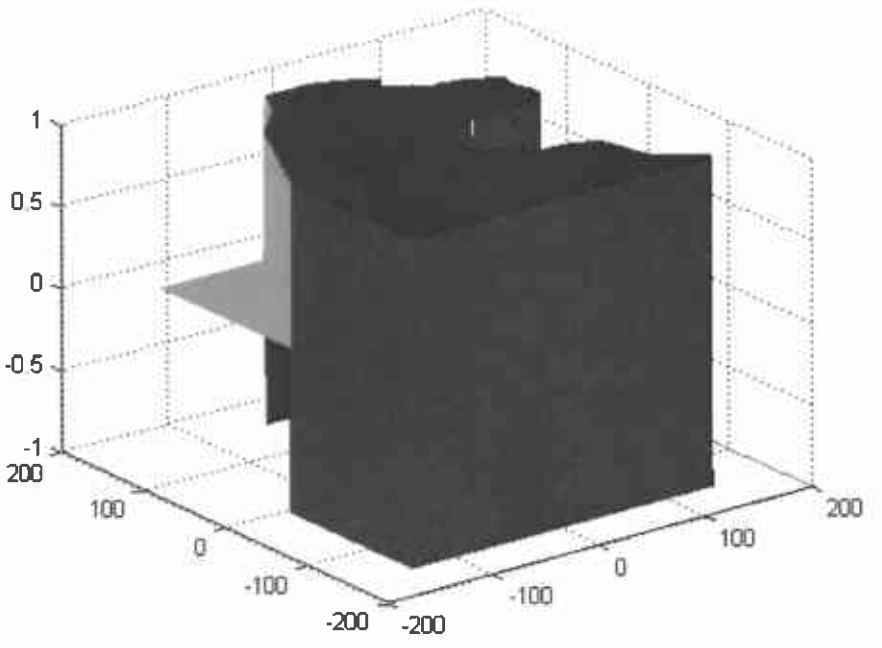


Figure A-3: Flow values for the different zones from the MODFLOW boundaries.

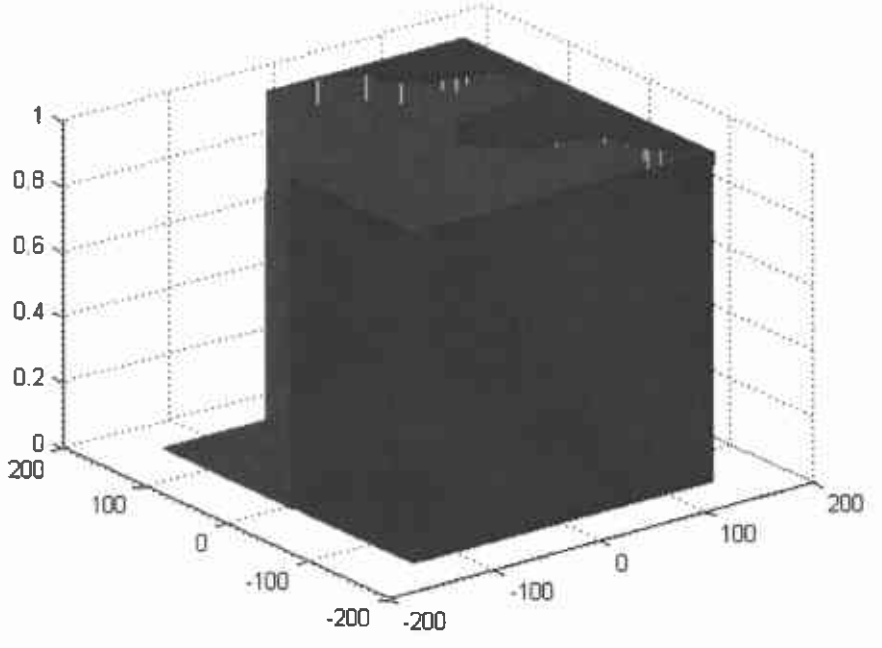


Figure A-4: Masking parameters created from flow boundaries.

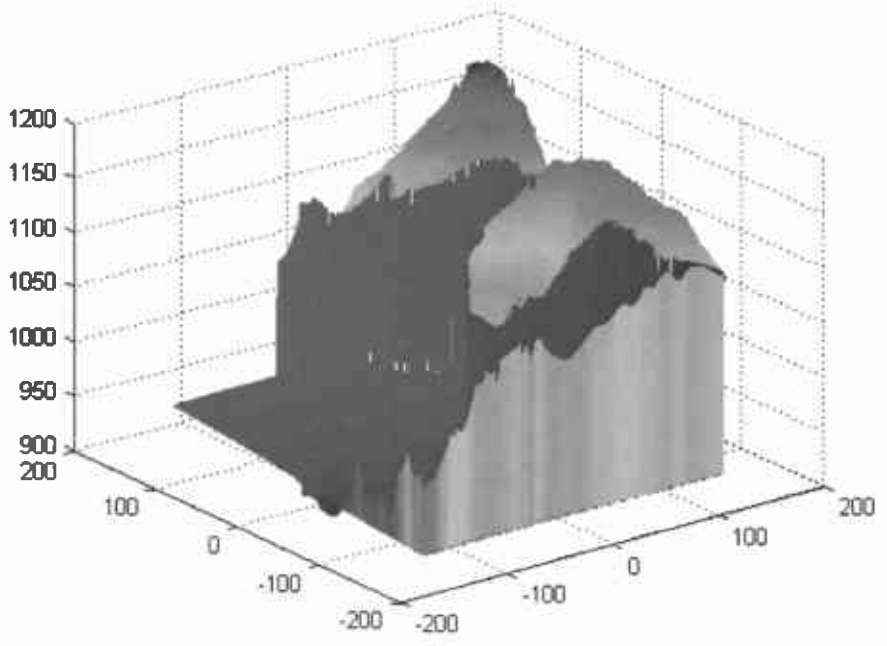


Figure A-5: Combined parametric and surface elevation values.

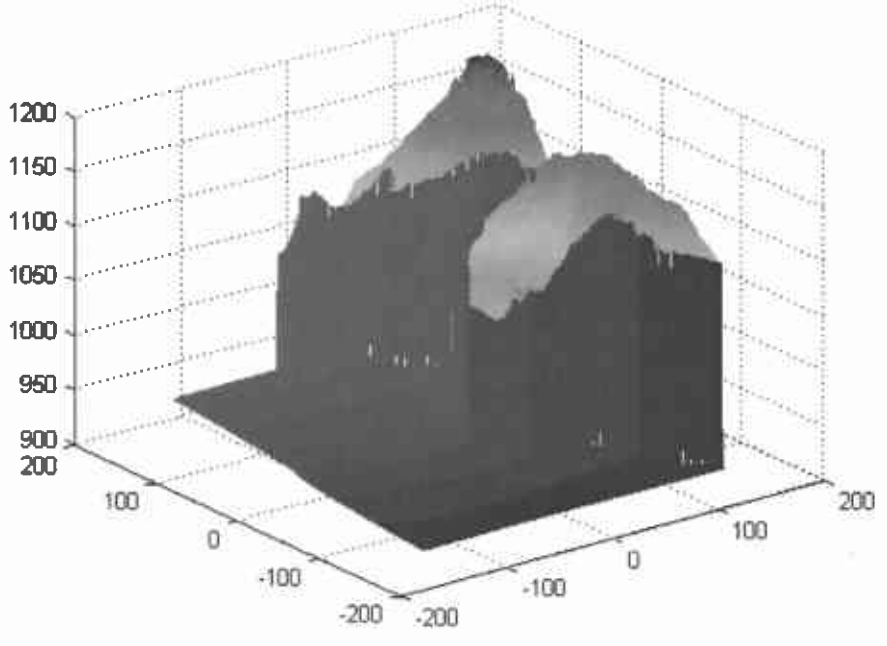


Figure A-6: Final initial head field with boundary conditions correct.

Appendix B: Routine Calculation – Creation of Transmissivity Variogram

A total of 65 wells provide some information on transmissivity in the Culebra within or near the model domain. Two wells, IMC-461 and SNL-6, provided only qualitative information on $\log_{10} T$ at the time of this analysis and were not used in this analysis. $\log_{10} T$ (m^2/s) measurements are available at 63 of these well, and 62 of those T values were used in the analysis. Well SNL-15 has a $\log_{10} T$ value, -12.9, that is more than 5 orders of magnitude below that of the next lowest T measurement and this well was removed from the variogram analysis as an outlier. Removal of this well from the analysis was based on the very low T measurement and also well SNL-15 being located in the geologically defined low-T zone to the east of the WIPP site.

For each of the 62 $\log_{10} T$ values used in this analysis, the well name, depth to Culebra and the easting (X) and northing (Y) coordinates are also given in UTM. These data are contained in the same file used for Task 5 of this AP. These data are used to create variograms of the $\log_{10} T$ values in the VarioWin COTS software package. The variogram model and parameters are then used as input to the T-field optimization within PEST 9.11.

Several preprocessing steps were necessary prior to the calculations in VarioWin. The input files listed here are all available in CVS under the Analysis/DataFiles module. These were accomplished within the `T_Wells_UTMNAD27.xls` file in additional worksheets added to the original worksheet containing the data listing:

1) In order to render the data set to be more amenable to the variogram calculations in VarioWin, the data coordinates were translated from the original coordinates by subtracting a value of 600,000 from each Easting (UTM X) coordinate and a value of 3,500,000 from each Northing (UTM Y) coordinate. This translation was accomplished in the `T_Wells_UTMNAD27.xls` file in the "PreVario" worksheet and the locations of the well data both prior and after the translation are shown in Figure B-1. This translation does not affect the shape of the variogram or the fit of the variogram model and there was no need to back-transform these coordinates after the variogram calculations.
2) VarioWin requires a tab delimited text file as input. This file contains 3 columns: the translated X and Y UTM coordinates and the data values in $\log_{10} T$ space. This file is organized in the "data_file" worksheet within the `T_Wells_UTMNAD27.xls` file. Additionally, the blank rows in the original worksheet used for organizational purposes are removed from this worksheet. The 3 columns in the "data_file" worksheet are saved as a tab delimited text file and the 5 line header required by VarioWin is added to this file. This file is `Analysis/DataFiles/T_wells.dat` in CVS.

The first step in the variogram calculation and modeling was to try and identify any statistical anisotropy in the spatial correlation of the data. This check was completed through calculation of a "variogram map" (Figure B-2). The variogram map provides a top-down view of the variogram values radiating

outward to increasing lag spacings in all directions from the center point of zero lag spacing. The lag spacing used for this calculation is 2500m in the X and Y directions. Figure B-2 shows some indication of stronger correlation in the NE-SW direction at short lag spacings, but further analysis with directional variograms showed that these differences were insignificant.

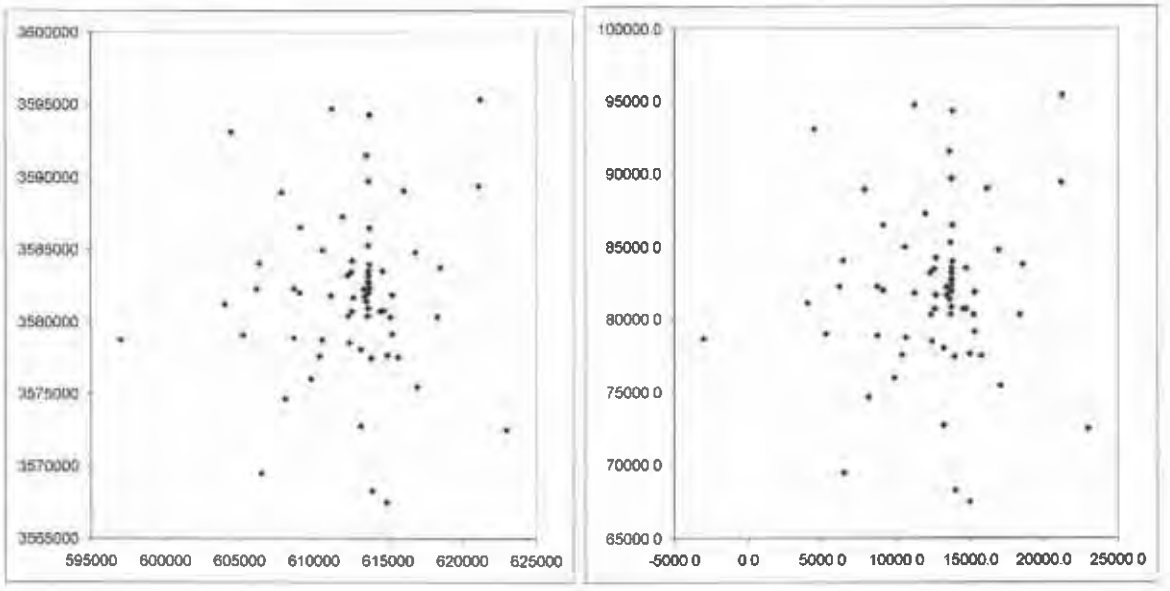


Figure B-1: Locations of wells used in the calculation of the log10 T variograms. The original and translated coordinate systems are shown in the left and right images, respectively.

For these analyses an omnidirectional variogram is calculated and fit with an exponential model. The variogram parameters are given in Table B-1 and Figure B-3 shows the experimental and model variograms. The emphasis in fitting the variogram model was on matching the experimental points with the smallest lag spacings and to constrain the model to the theoretical sill that is the variance of the data set (horizontal dashed line in Figure B-3). As expected, the variogram model showed a lower nugget and a larger range relative to the residual variogram used in the previous CRA calculations. The parameters in Table B-1 were used directly as input to the PEST PPK2FAC utility.

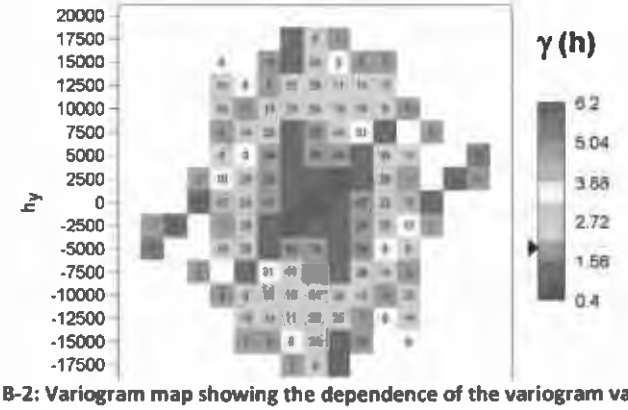


Figure B-2: Variogram map showing the dependence of the variogram values on the orientation. The color scale indicates the variogram value and the black triangle identifies the theoretical sill (variance of the data set).

Table B-1: Variogram parameters for isotropic fit. Omnidirectional variogram calculated with a lag spacing of 1500m.

Parameter	Value
Model Type	Exponential
Nugget	0.02 $(\log10\ T)^2$
Sill	1.95 $(\log10\ T)^2$
Range	9500 meters

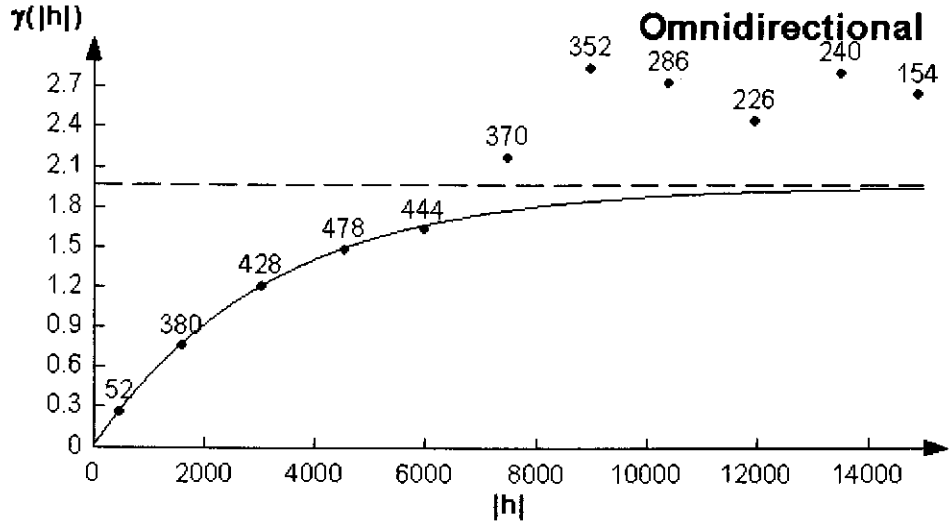


Figure B-3: Experimental variogram (black dots) and exponential model (black line).

[THIS PAGE INTENTIONALLY LEFT BLANK]

Appendix C: Results – Modeled Steady-State Head

This appendix contains the data and results of the steady-state modeled head on the complete set of 200 fields and the final set of 100 fields. The figures that are presented represent only the results of the 100 best fields. The data tables and summaries will provide details for all fields. The field measurements for the 44 wells used are from (Johnson, 2009) and are presented in Table C-1.

Table C-1: Field freshwater head equivalent measurements.

Well Name	Head (m AMSL)	Well Name	Head (m AMSL)
C-2737	921.23	SNL-9	932.05
ERDA-9	924.88	SNL-10	931.54
H-2b2	929.62	SNL-12	915.24
H-3b2	918.68	SNL-13	918.19
H-4b	916.34	SNL-14	916.33
H-5b	939.12	SNL-15	1060
H-6b	936.44	SNL-16	918.68
H-7b1	914.58	SNL-17	916.78
H-9c	912.8	SNL-18	939.87
H-10c	922.02	SNL-19	937.58
H-11b4	917.09	USGS-4	911.11
H-12	916.53	WIPP-11	940.65
H-15	920.32	WIPP-13	939.78
H-17	916.24	WIPP-19	933.66
H-19b0	918.84	WIPP-25	937.57
IMC-461	928.95	WIPP-30	939.37
SNL-1	941.86	WQSP-1	938.28
SNL-2	937.65	WQSP-2	939.87
SNL-3	939.81	WQSP-3	936.43
SNL-5	938.59	WQSP-4	919.5
SNL-6	1110	WQSP-5	918.18
SNL-8	929.94	WQSP-6	921.96

In Figure C-1 through Figure C-6, the distribution of the 100 best results is compared to the measured values on a well-by-well basis, from lowest observed values to highest observed value. SNL-6 and SNL-15 are fixed, and are not presented. In Figure C-7 through Figure C-10, the one-to-one correspondence of expected value vs. modeled value for head is plotted to give an idea of the total fit of the final fields. In all the graphics, the modeled values are presented as gray markers and the observed values as red markers. The box-and-whisker markers include whiskers at the 2.5% and 97.5% values and a box around the central 50%; the median value is marked with a black line.

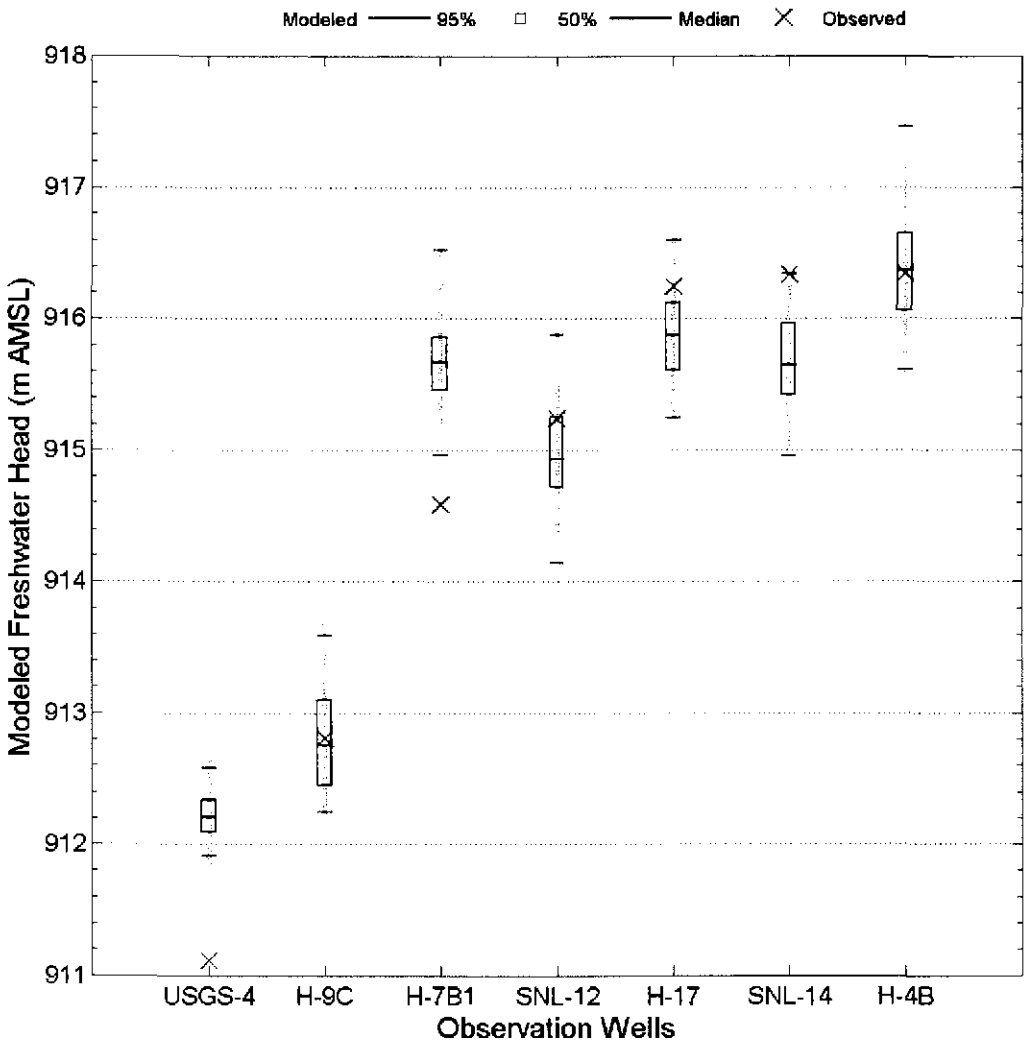


Figure C-1: First set of observed and measured steady-state head values presented by well name.

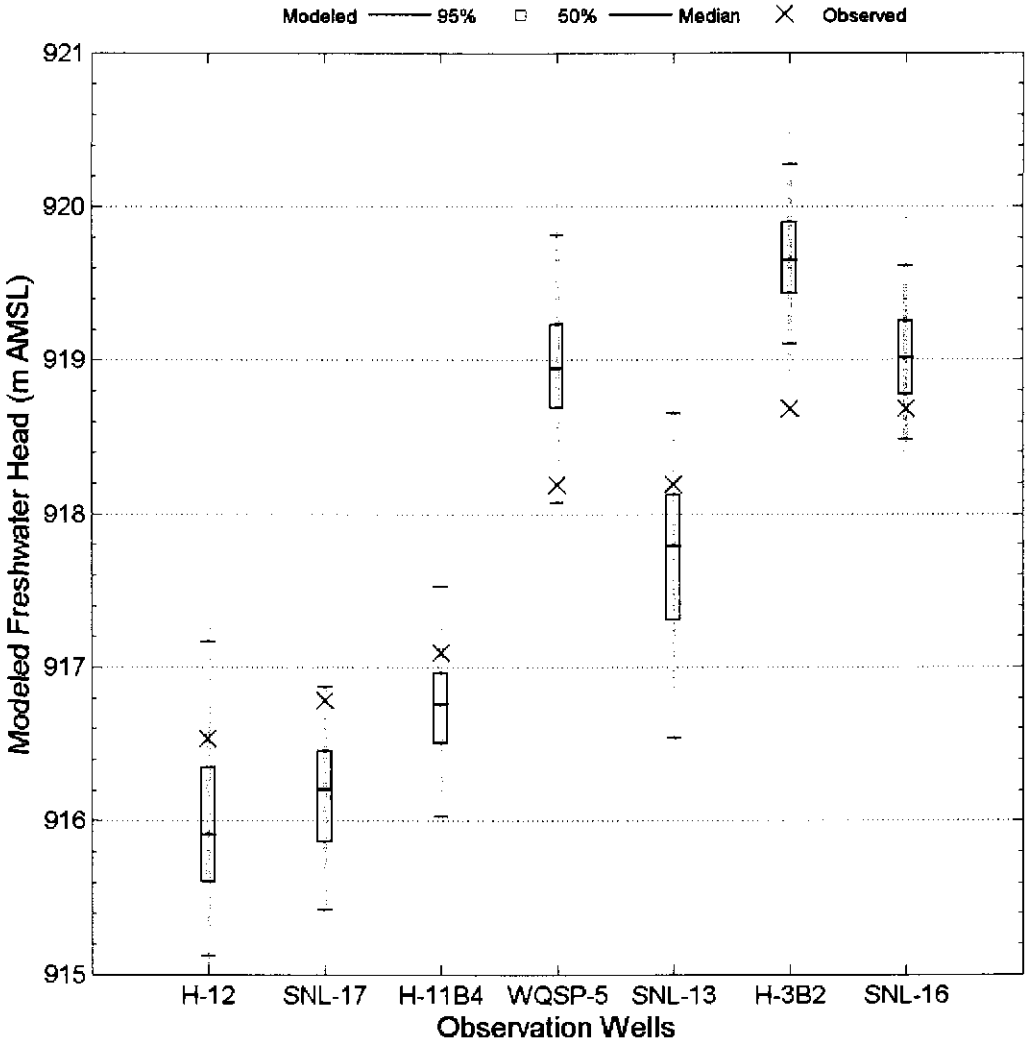


Figure C-2: **Second set of observed and measured steady-state head values presented by well name.**

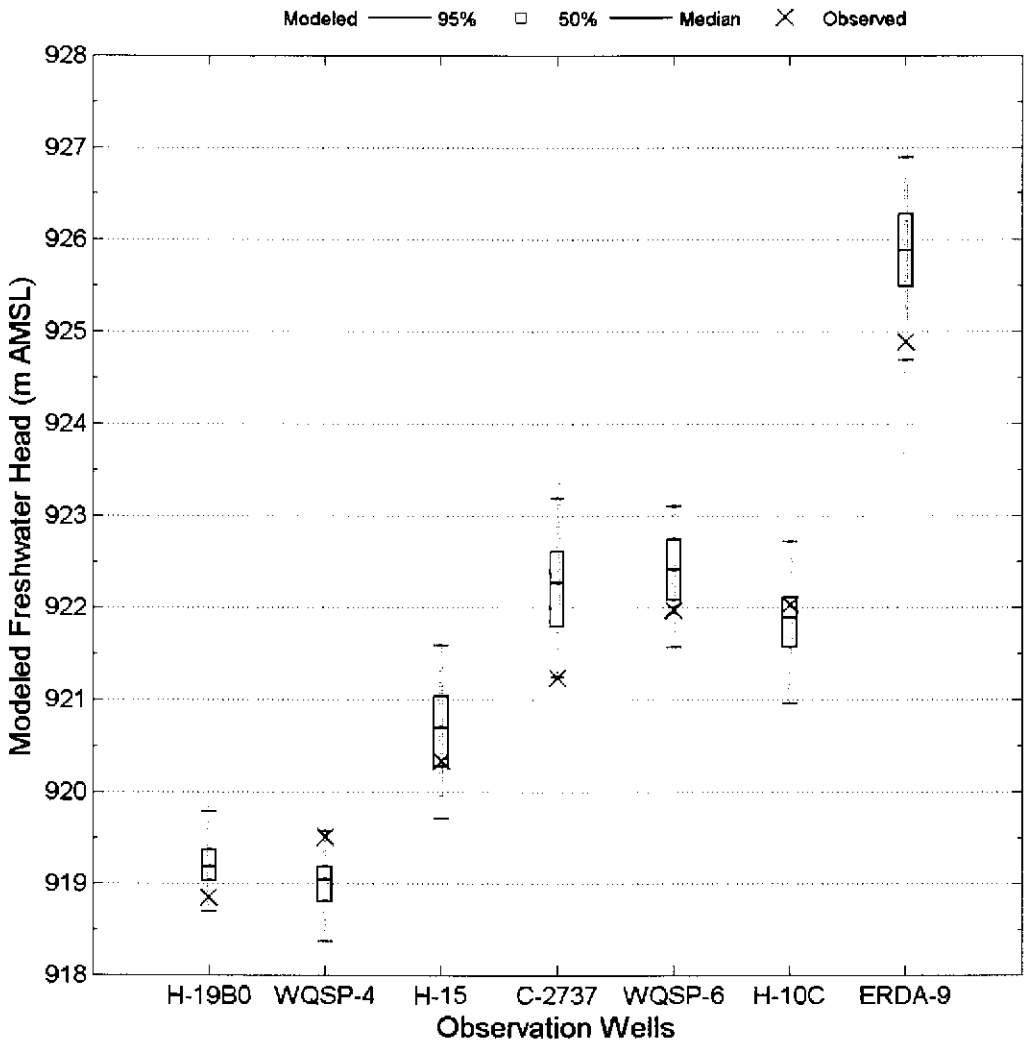


Figure C-3: Third set of observed and measured steady-state head values presented by well name.

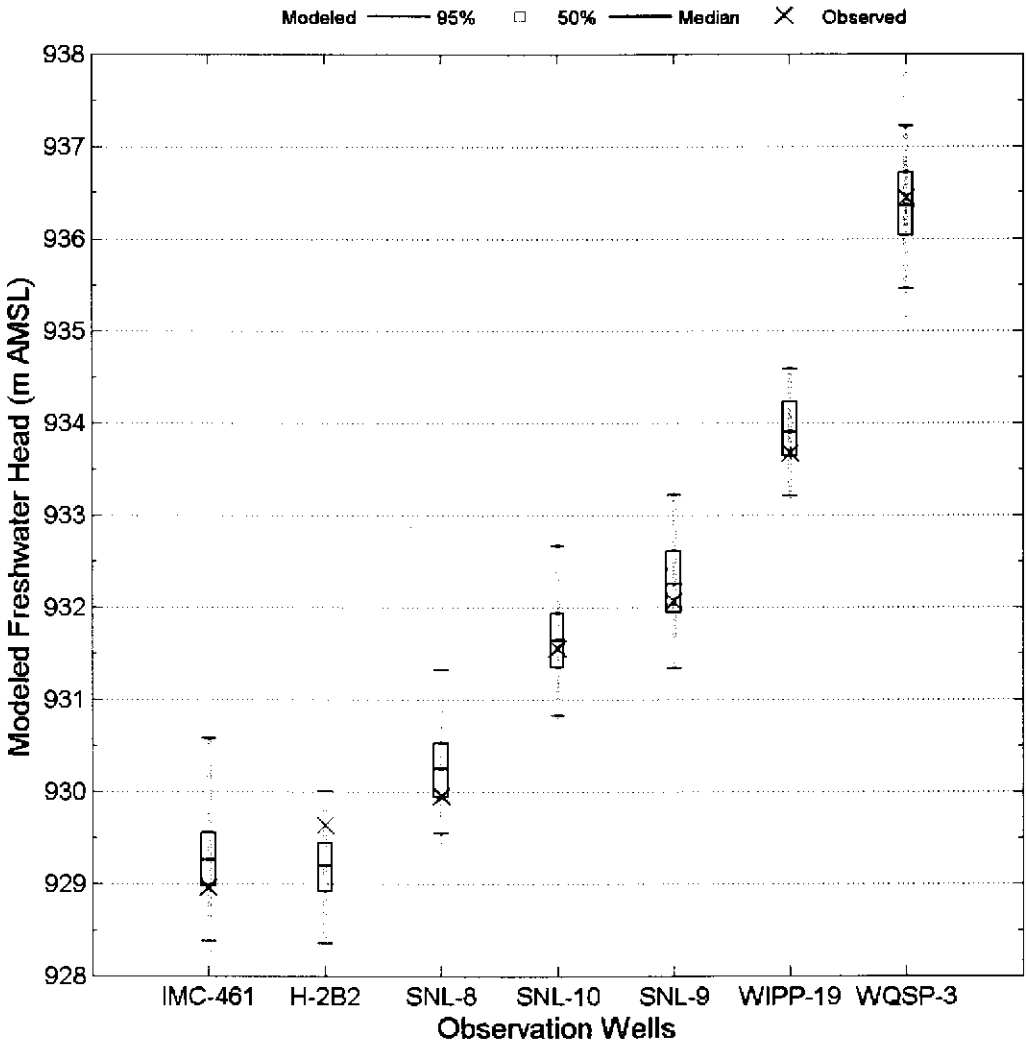


Figure C-4: Fourth set of observed and measured steady-state head values presented by well name.

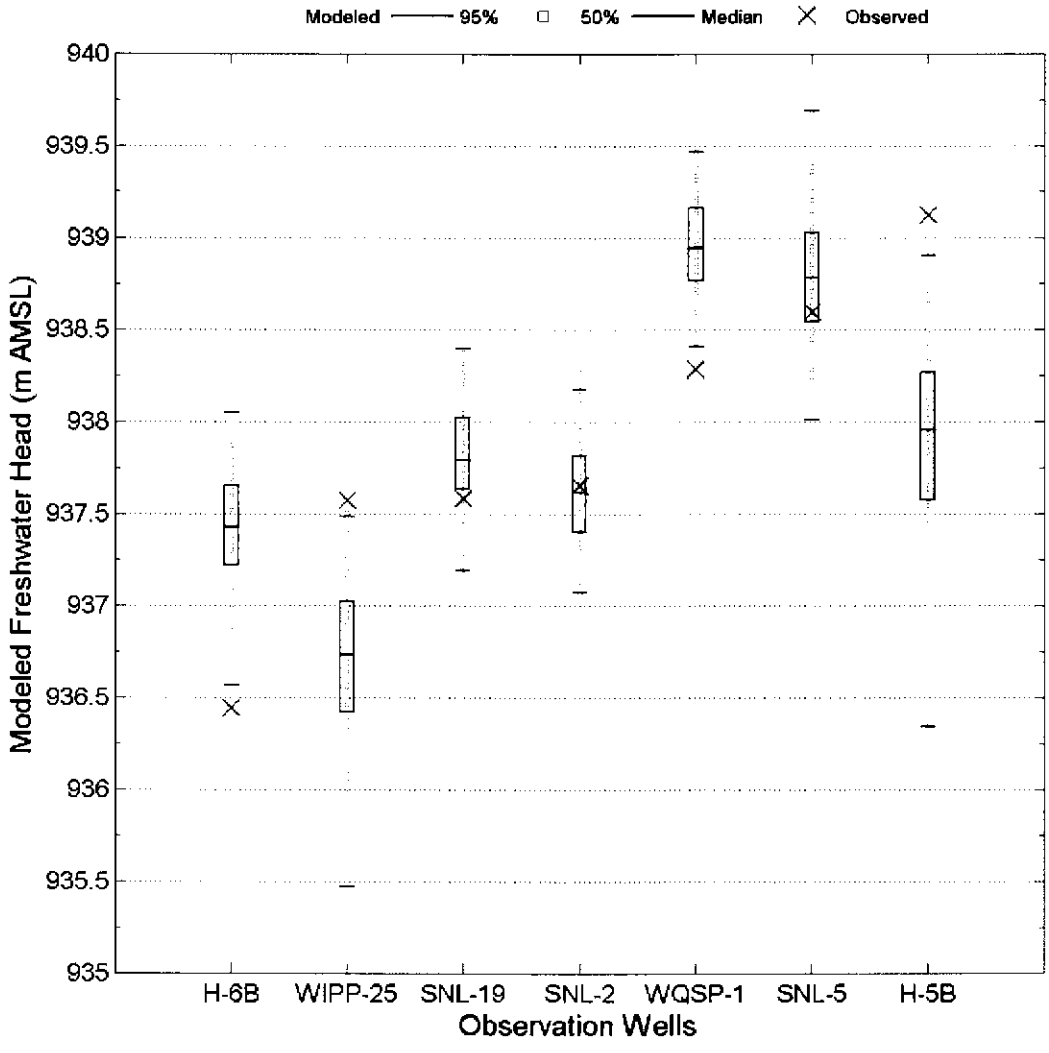


Figure C-5: Fifth set of observed and measured steady-state head values presented by well name.

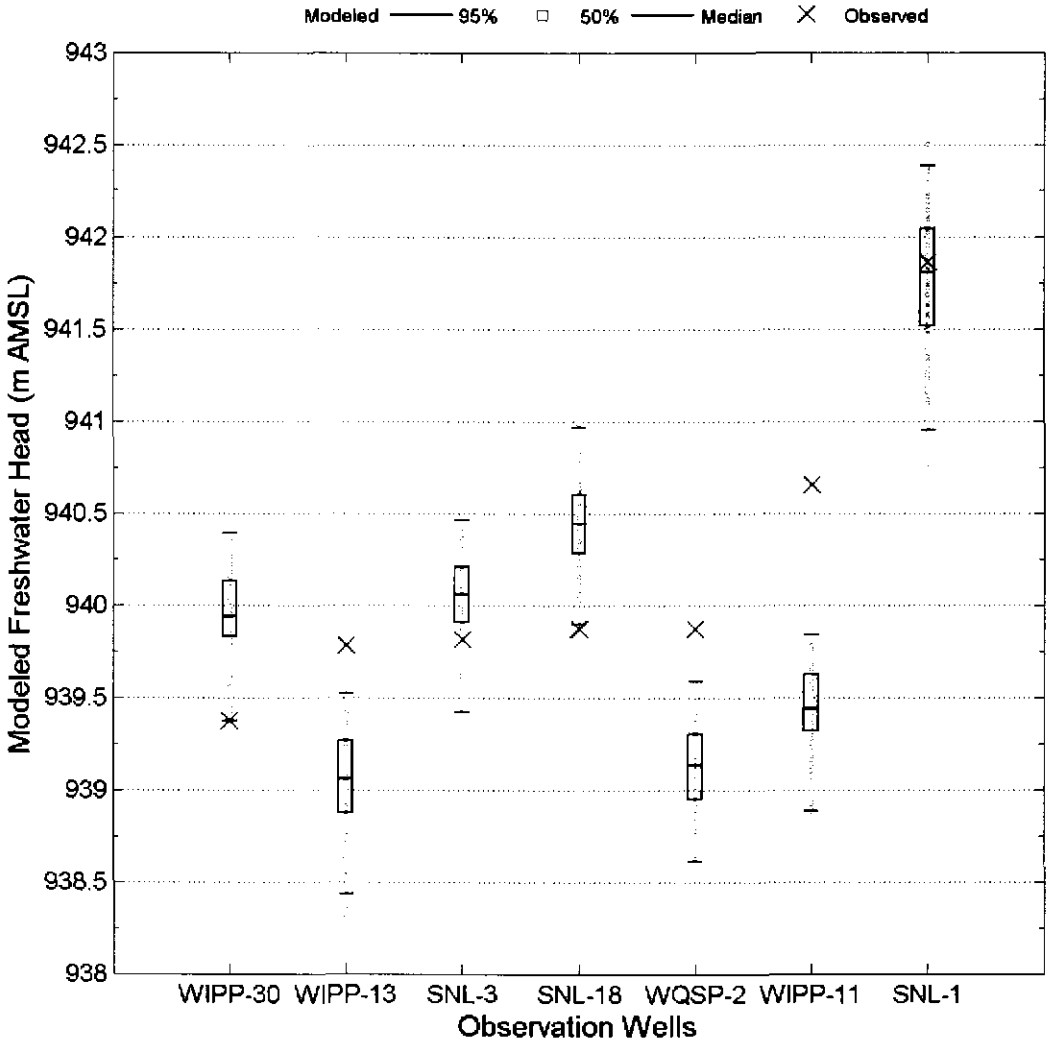


Figure C-6: Last set of observed and measured steady-state head values presented by well name.

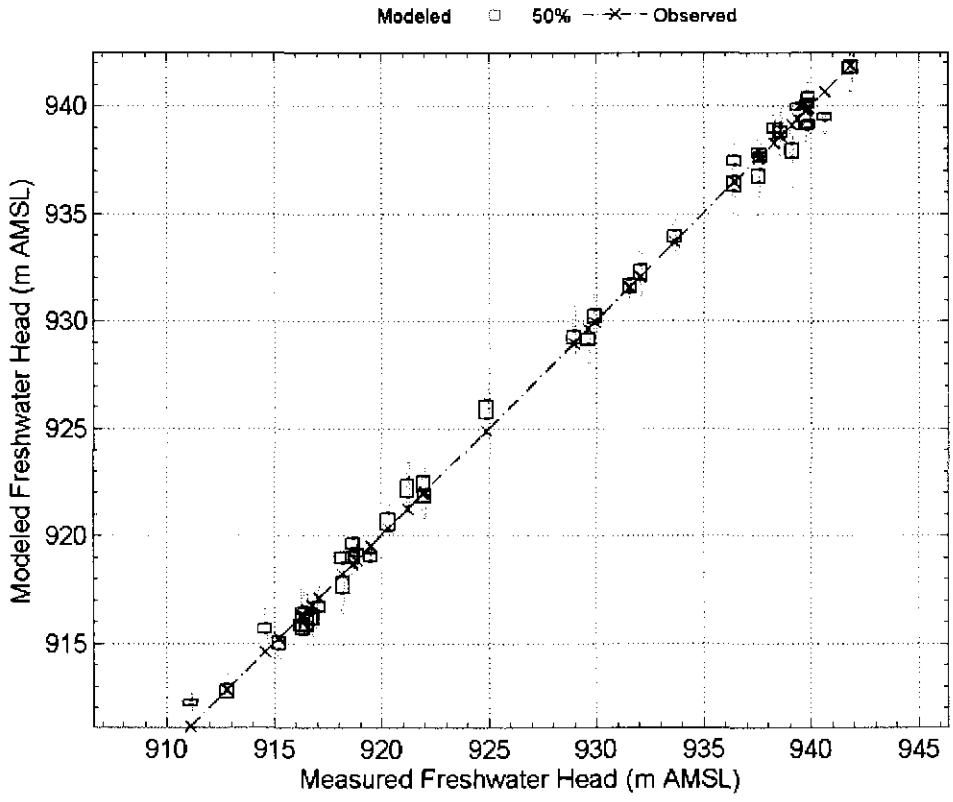


Figure C-7: The wells that defined the steady-state fit presented along a one-to-one axis, where the exact match would be on the axis. Measured values are presented is red "X" marks. The following three figures are subsets of this graph.

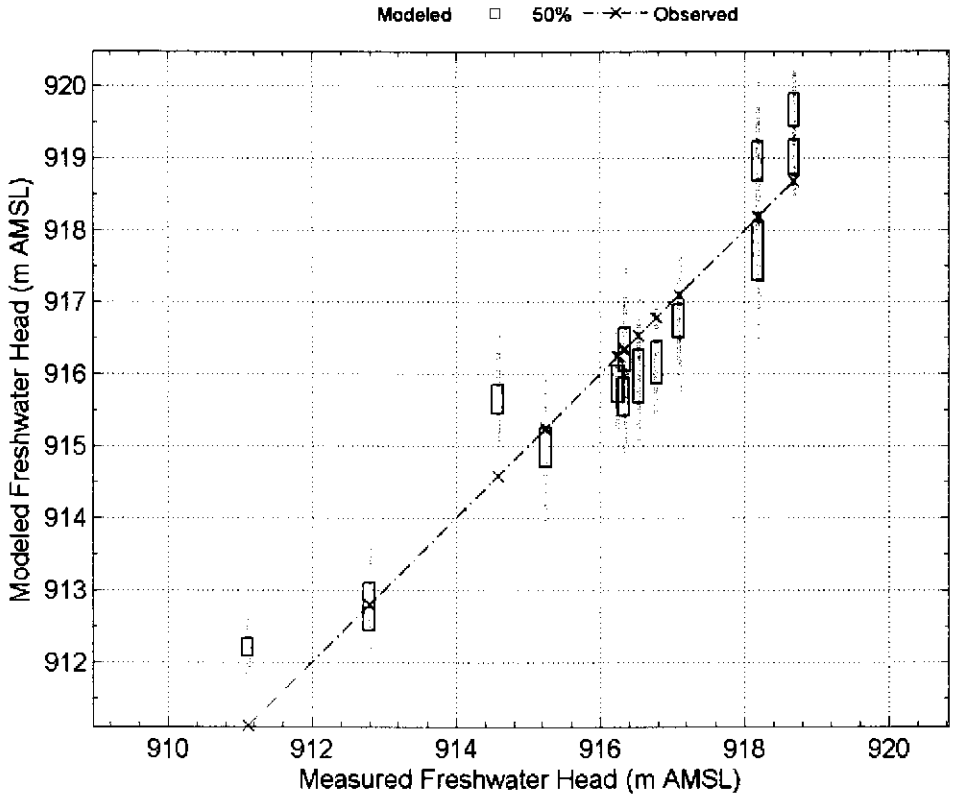


Figure C-8: The first 14 wells presented along the one-to-one line.

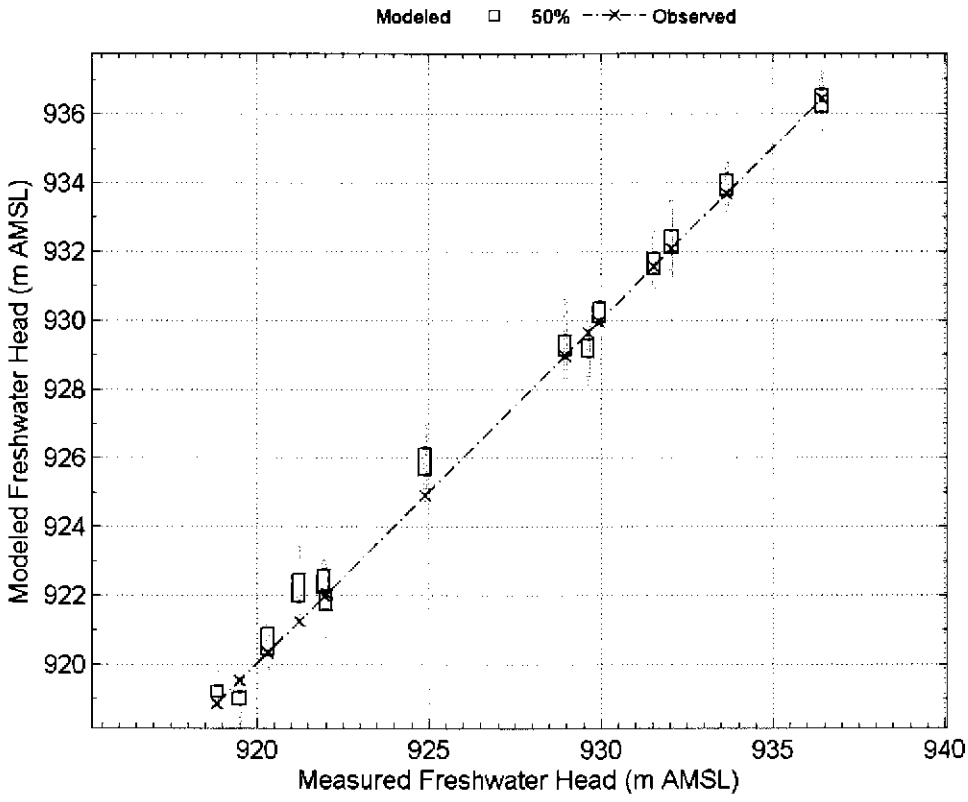


Figure C-9: The central 14 wells presented along the one-to-one line.

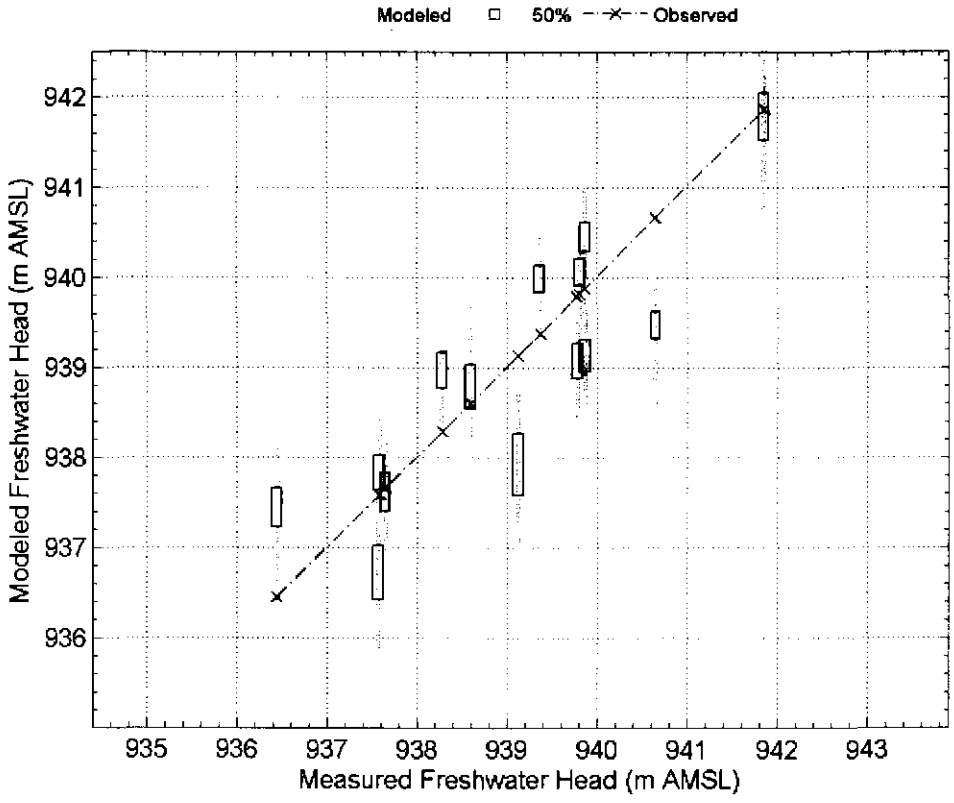


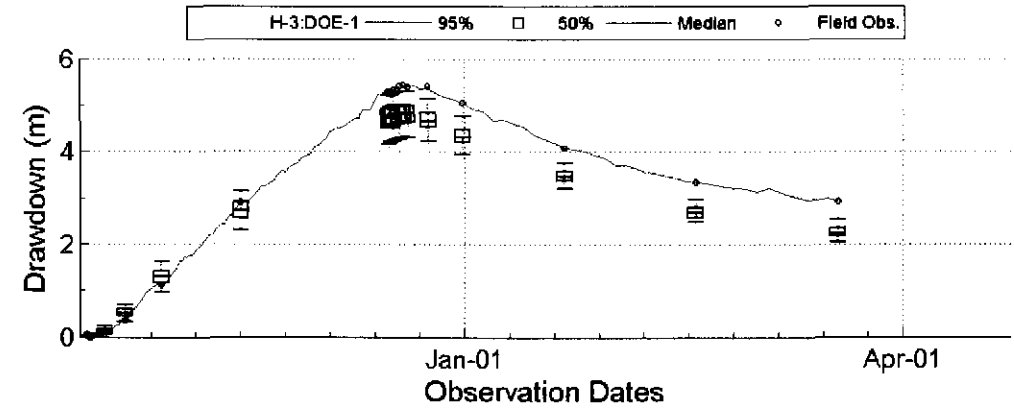
Figure C-10: The final 14 wells presented along the one-to-one line.

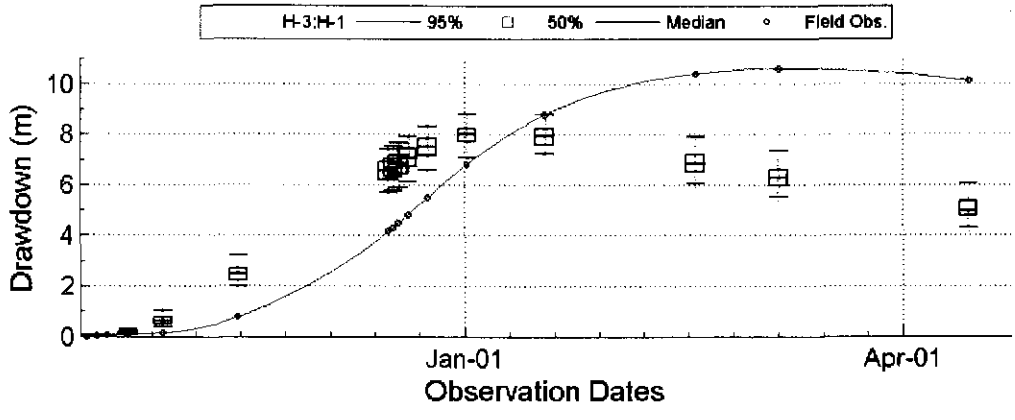
[THIS PAGE INTENTIONALLY LEFT BLANK]

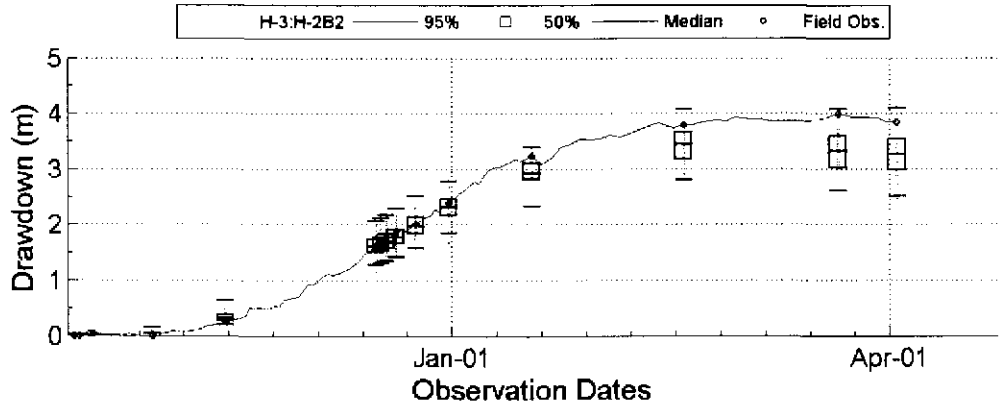
Appendix D: Results – Pumping Test Responses

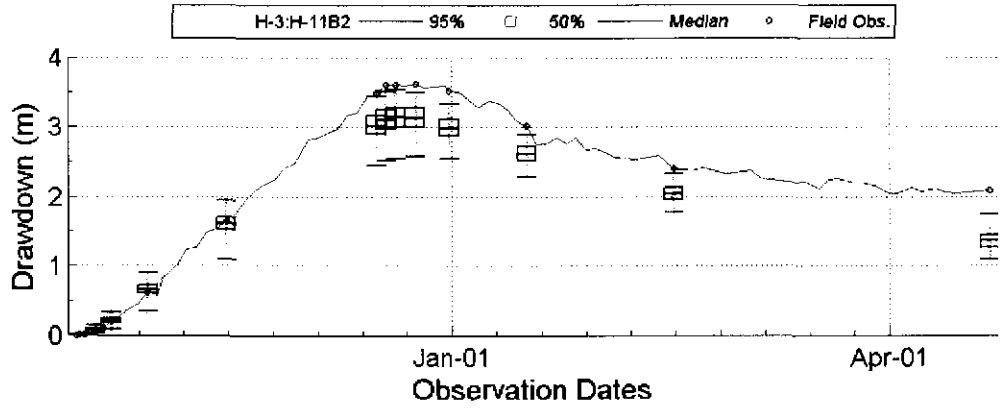
This appendix provides the pumping test results for the 100 selected fields in graphical form. Each observation well has the field measurements presented with the 100 modeled responses plotted with box-and-whisker plots at the observation points used in the PEST calibration.

D.1 Modeled Responses to the H-3 Pumping Test

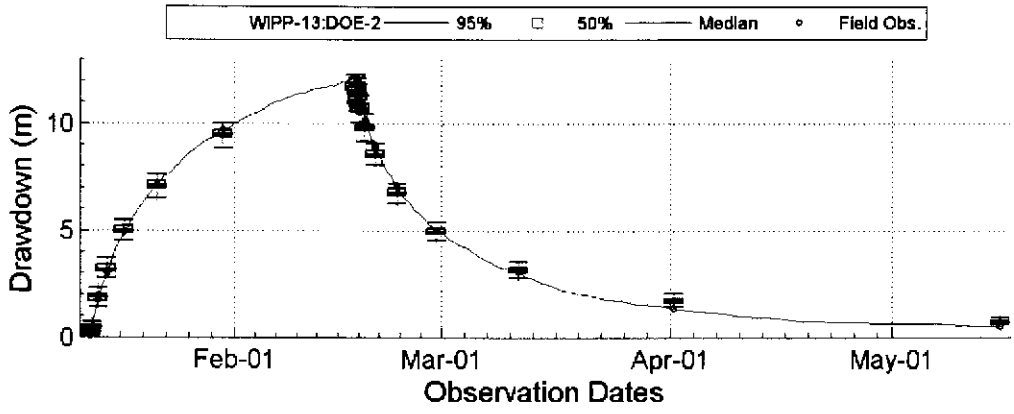


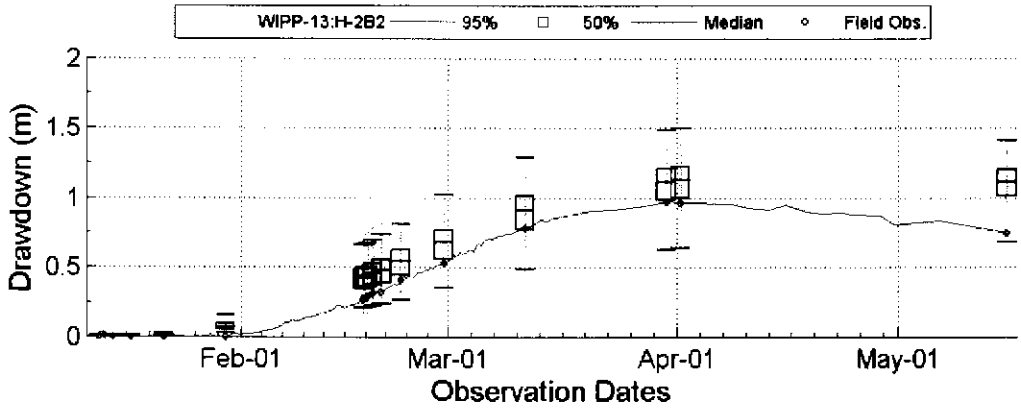


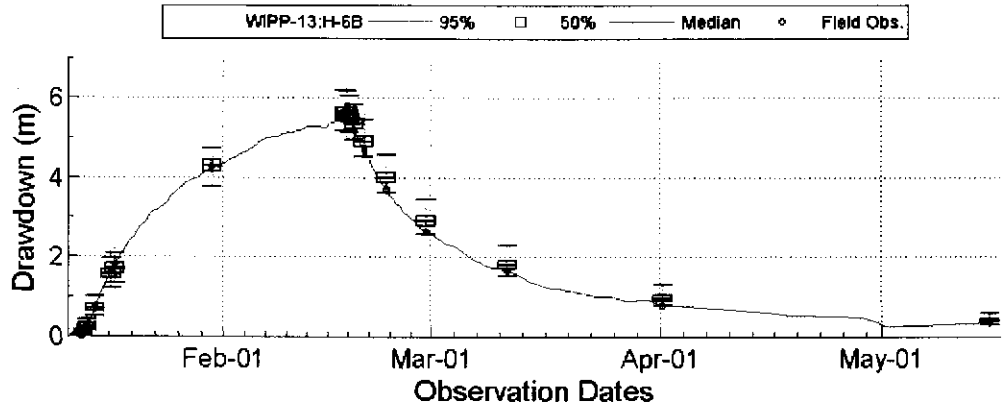


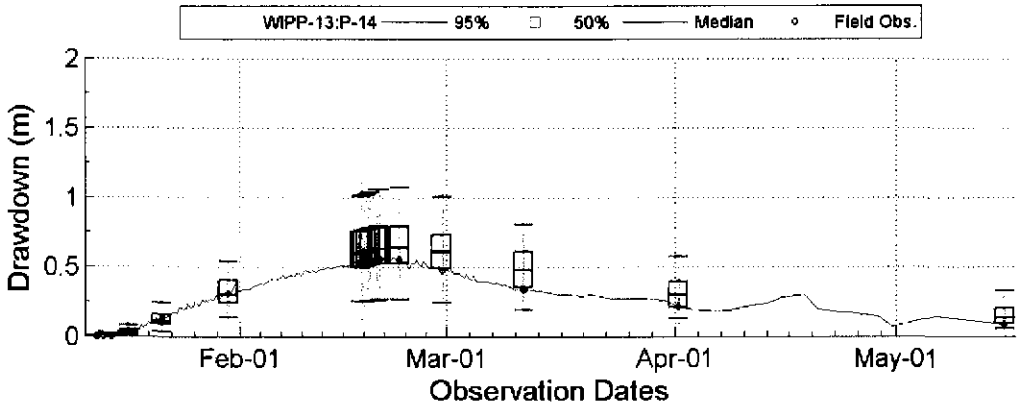


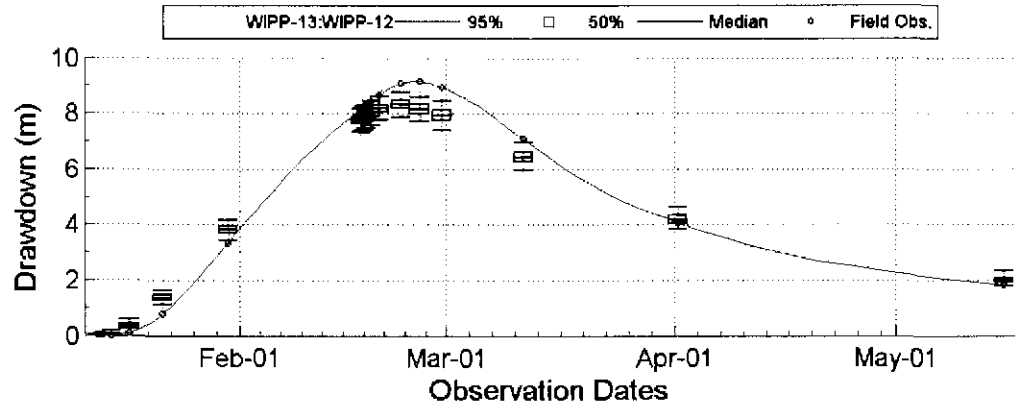
D.2 Modeled Responses to the WIPP-13 Pumping Test

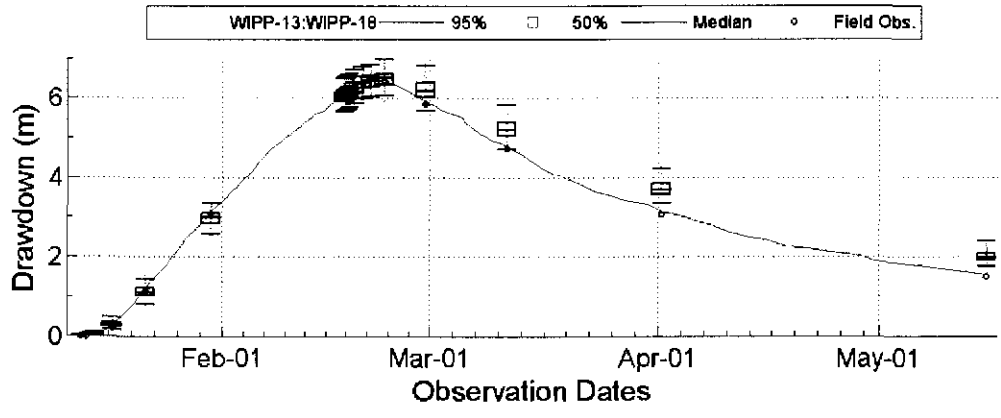


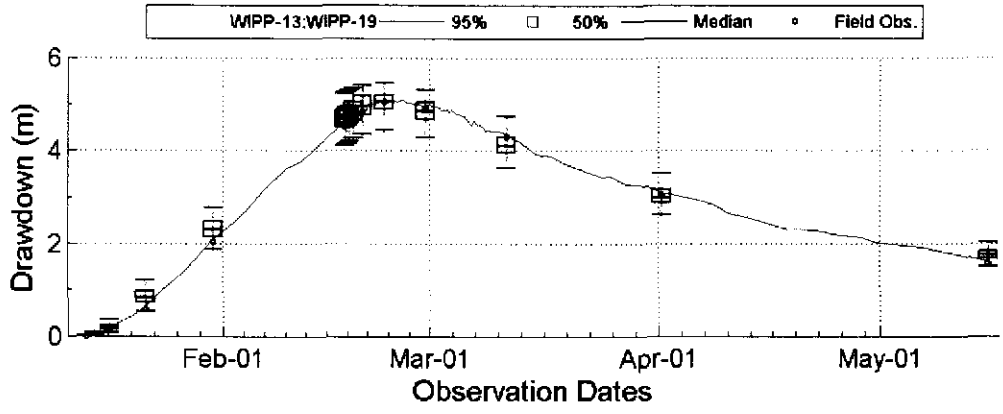


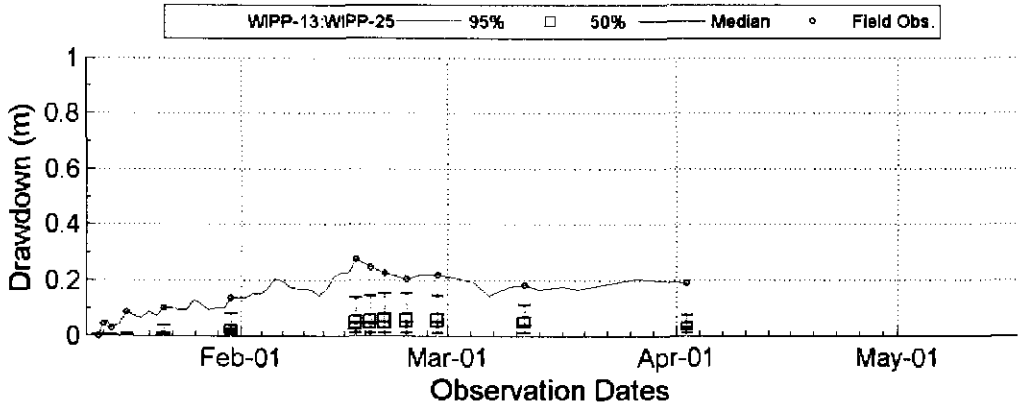


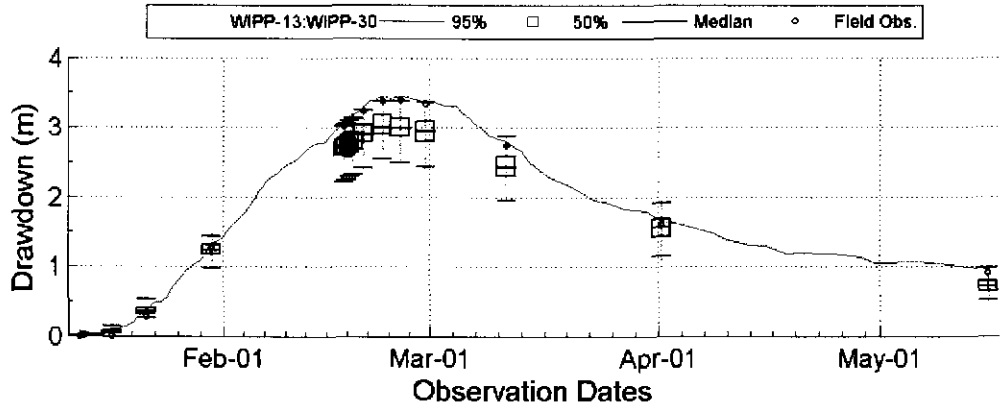




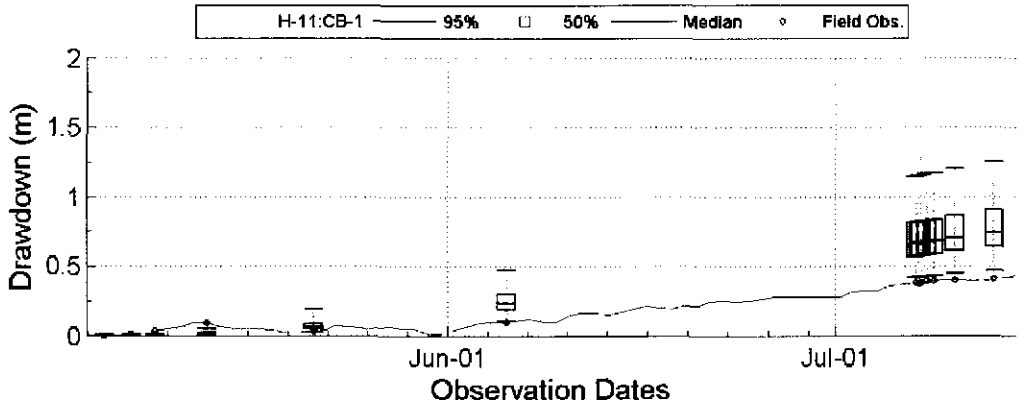


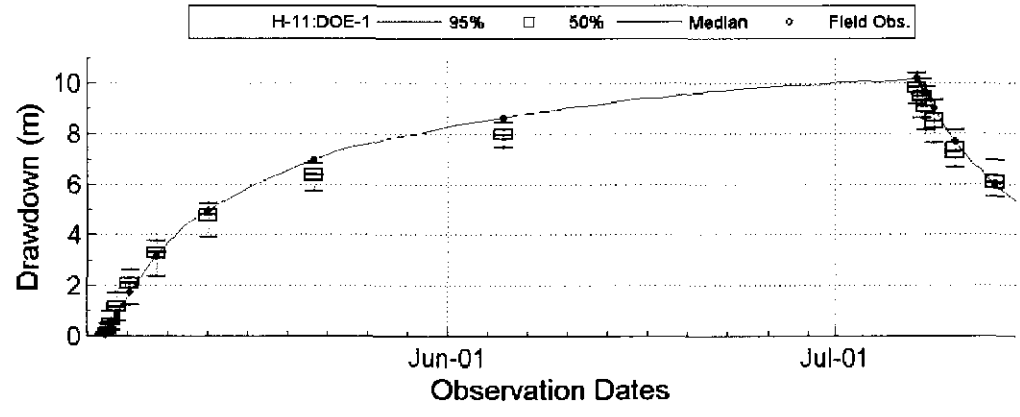


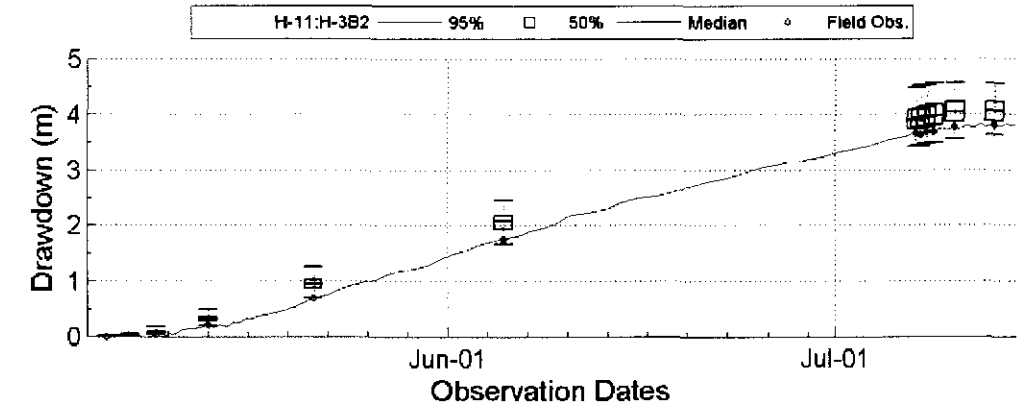


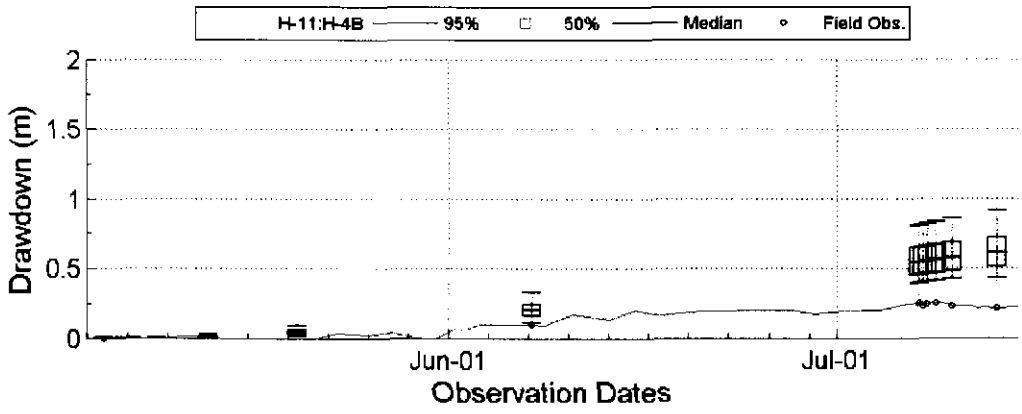


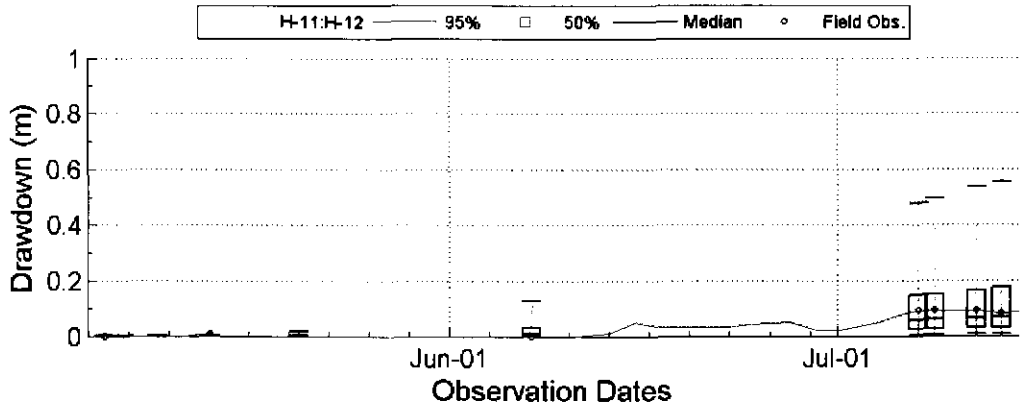
D.3 Modeled Responses to the H-11 (1988) Pumping Test

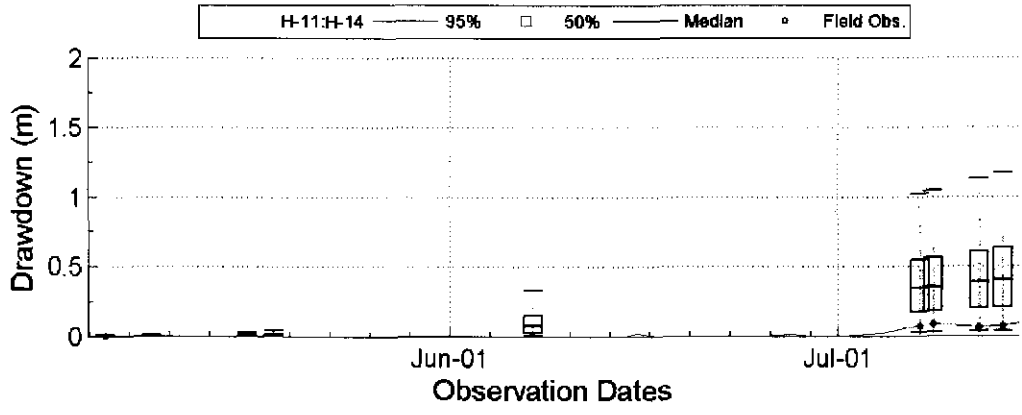


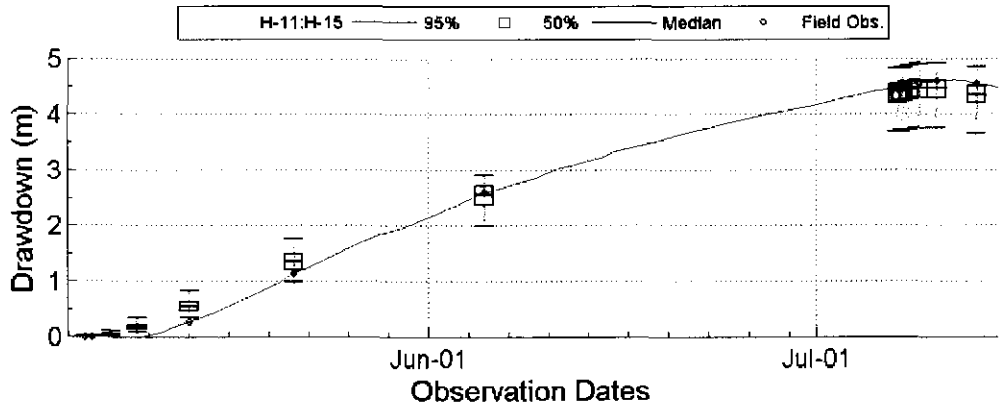


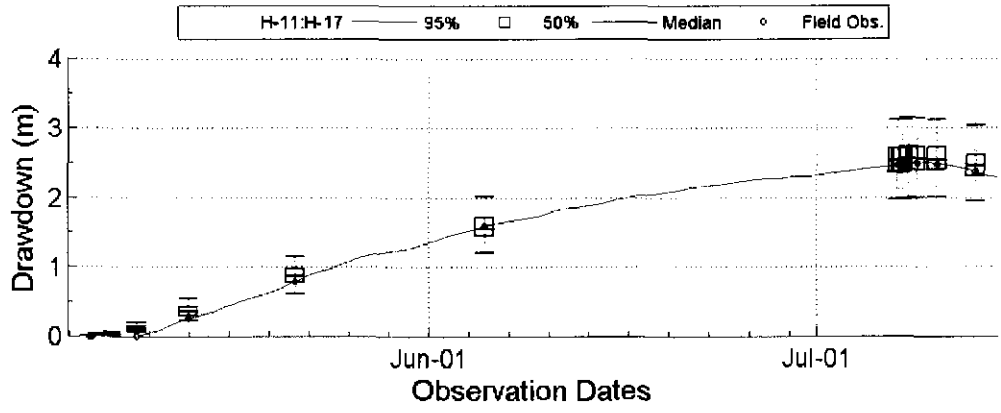


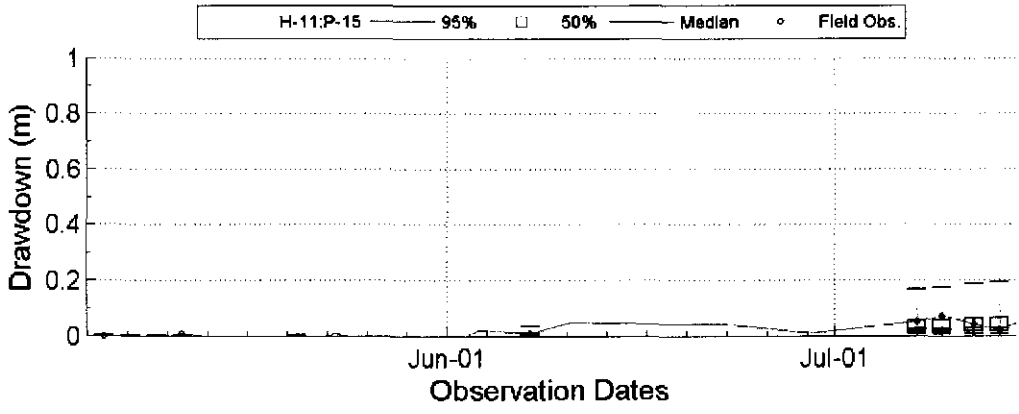


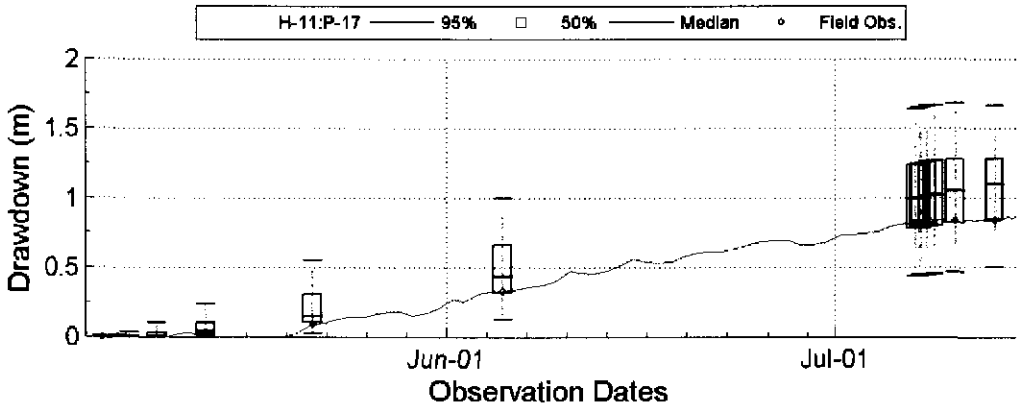




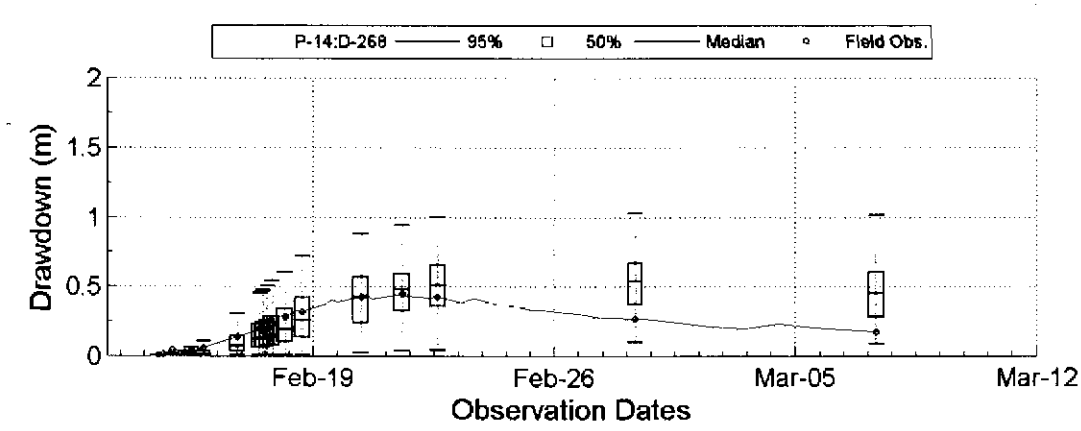


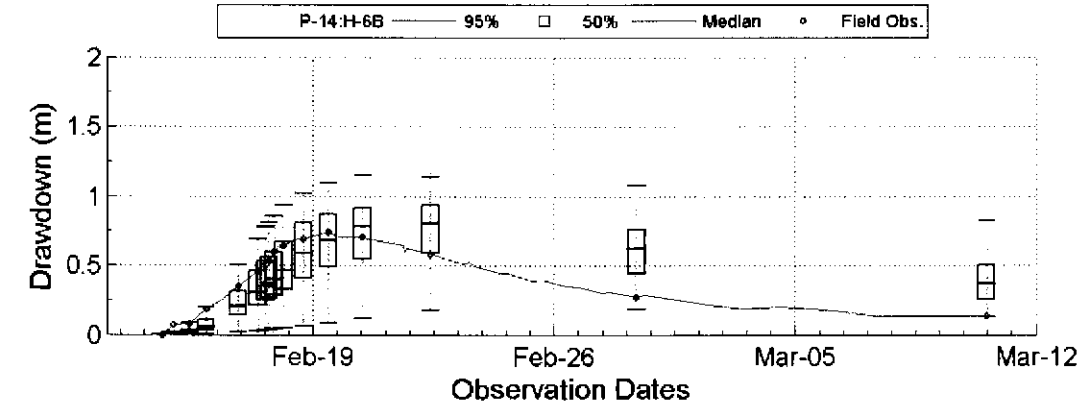


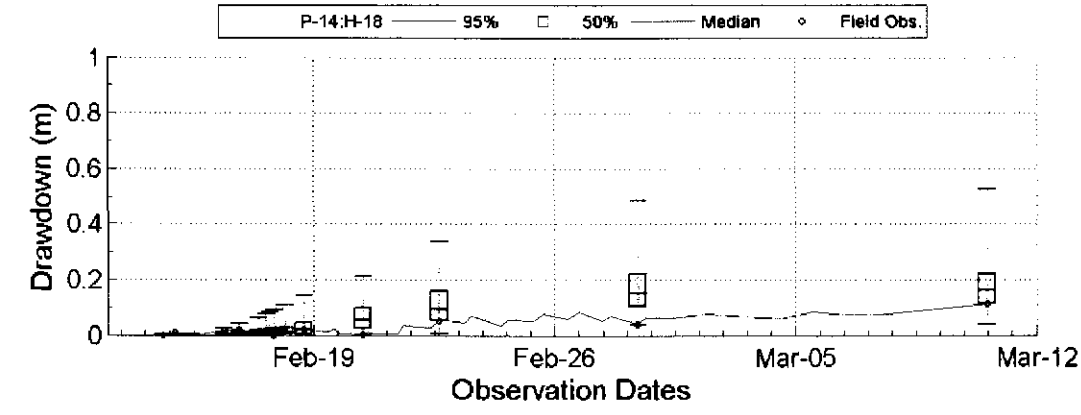


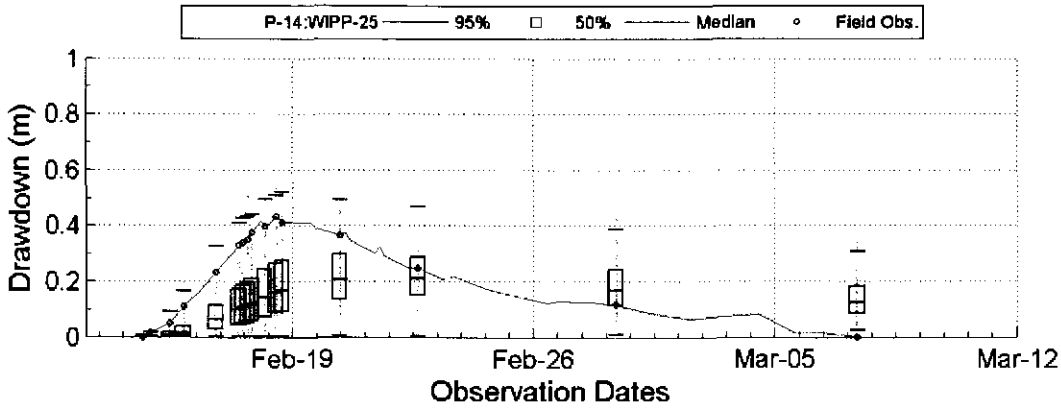


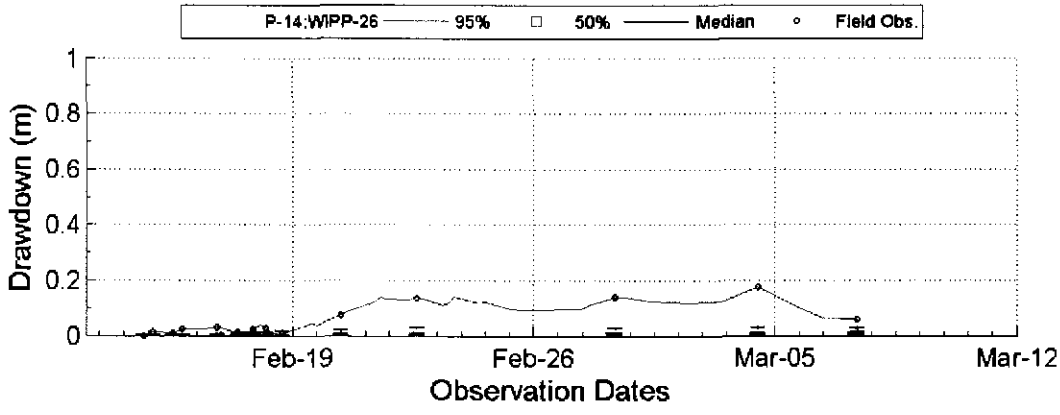
D.4 Modeled Responses to the P-14 Pumping Test



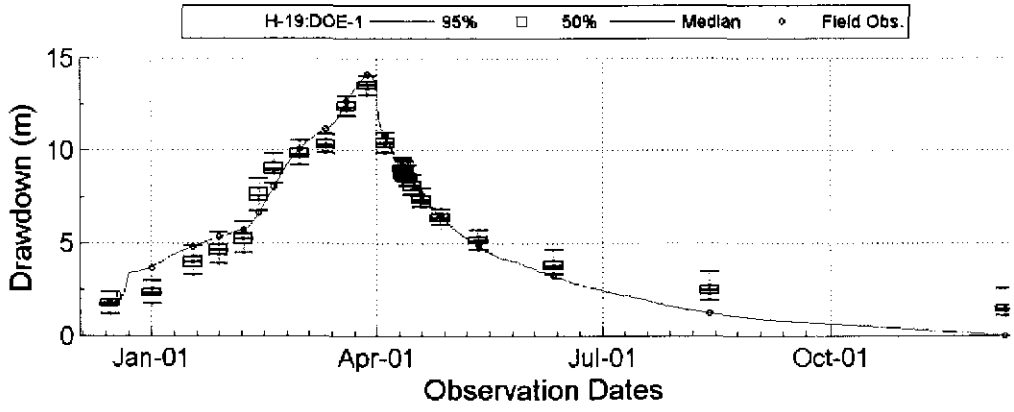


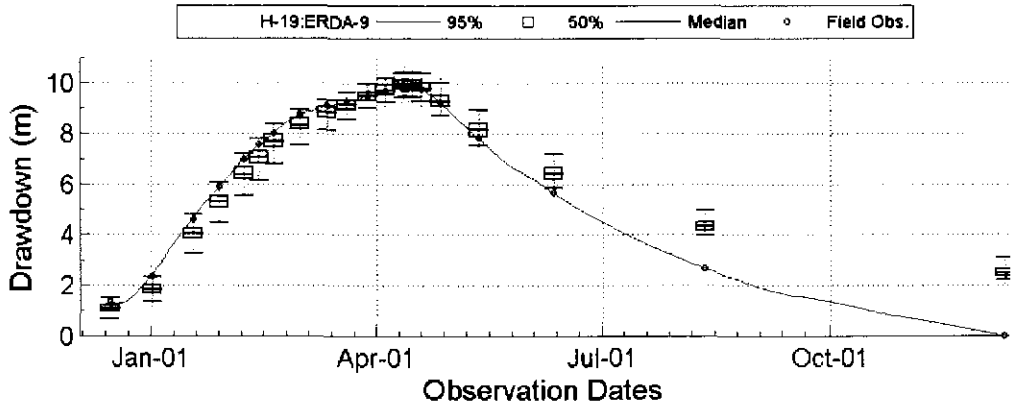


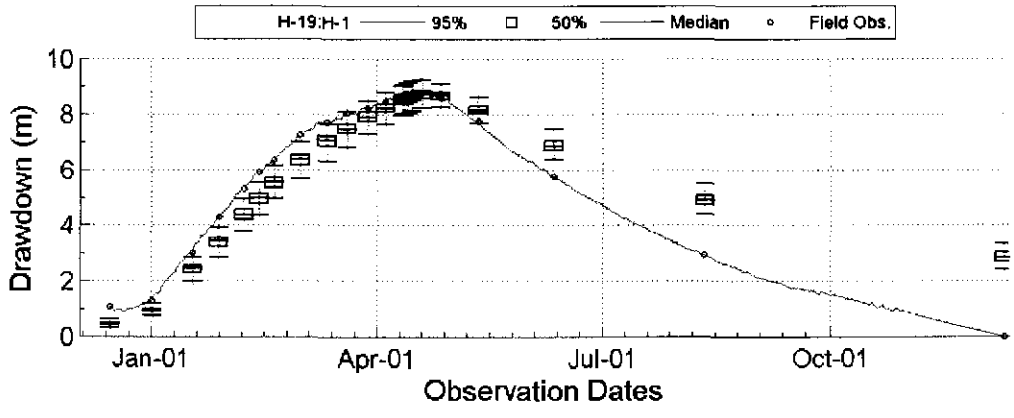


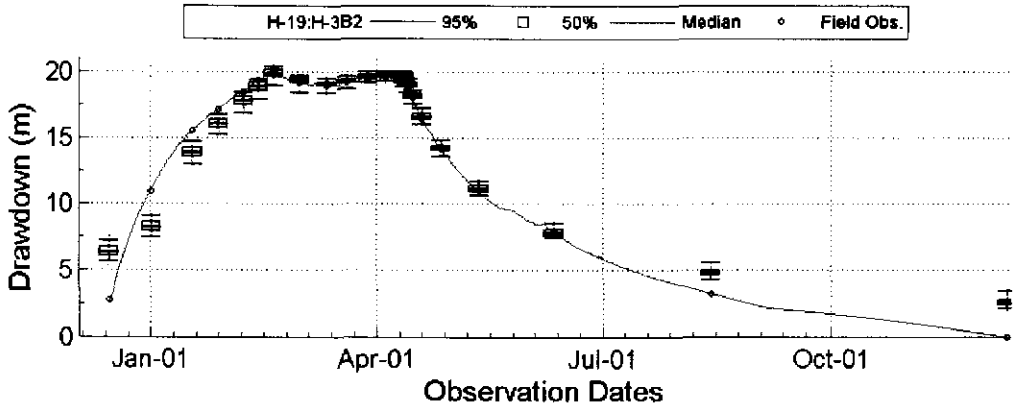


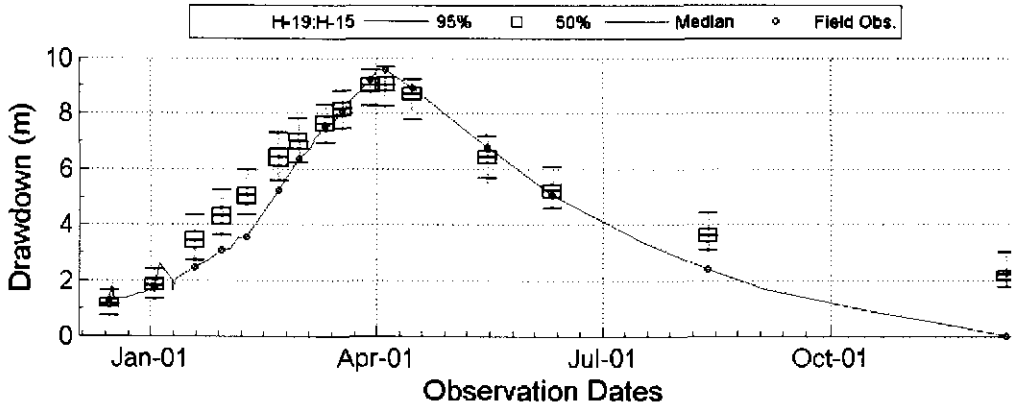
D.5 Modeled Responses to the H-19b0 Pumping Test

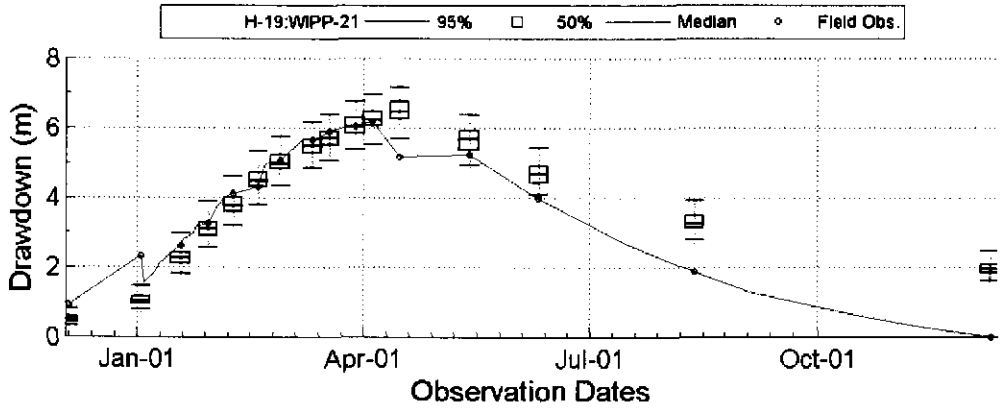


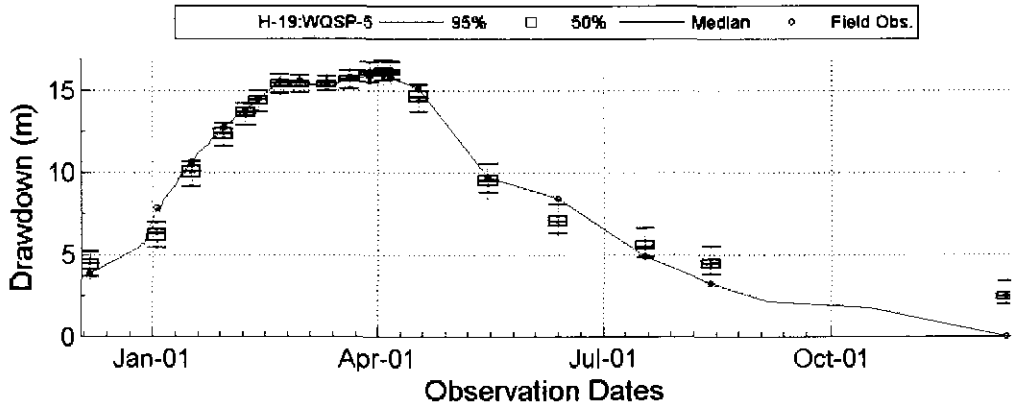




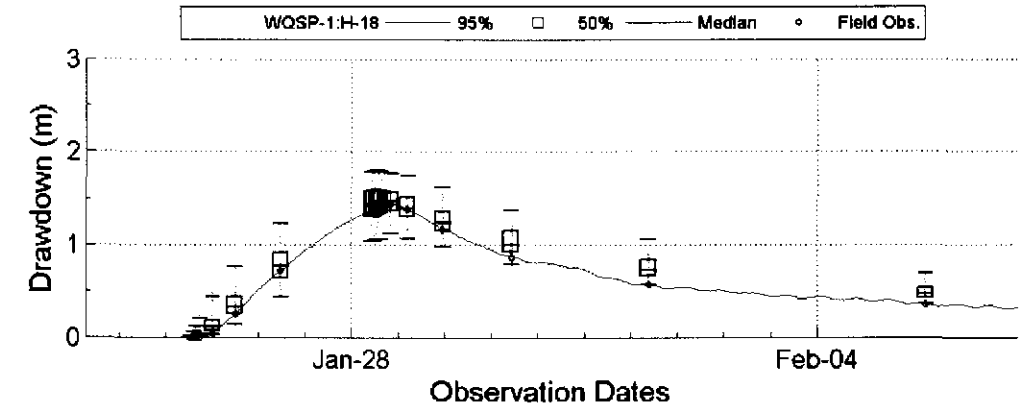


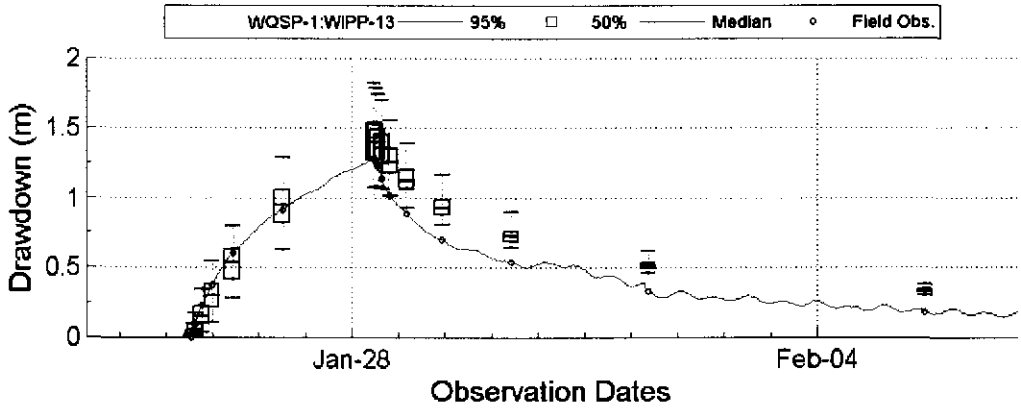




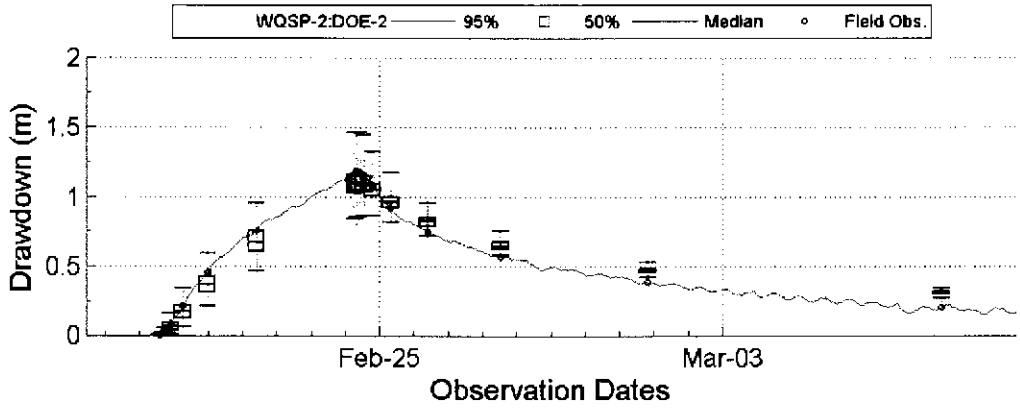


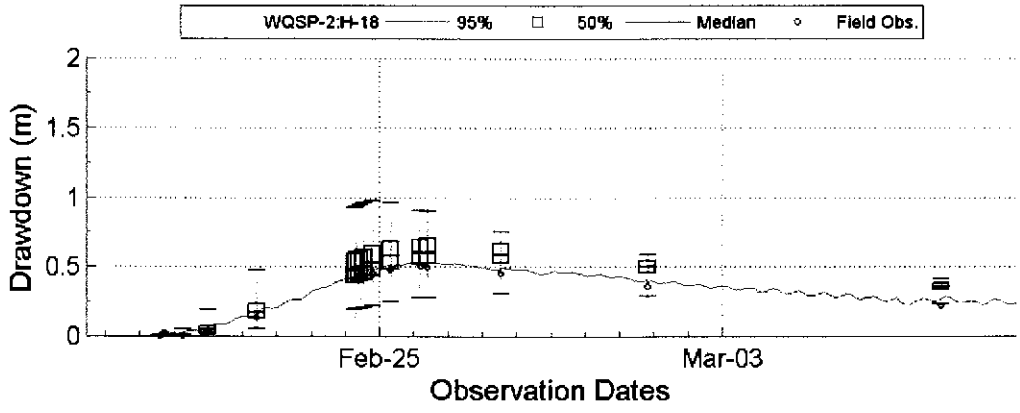
D.6 Modeled Responses to the WQSP-1 Pumping Test

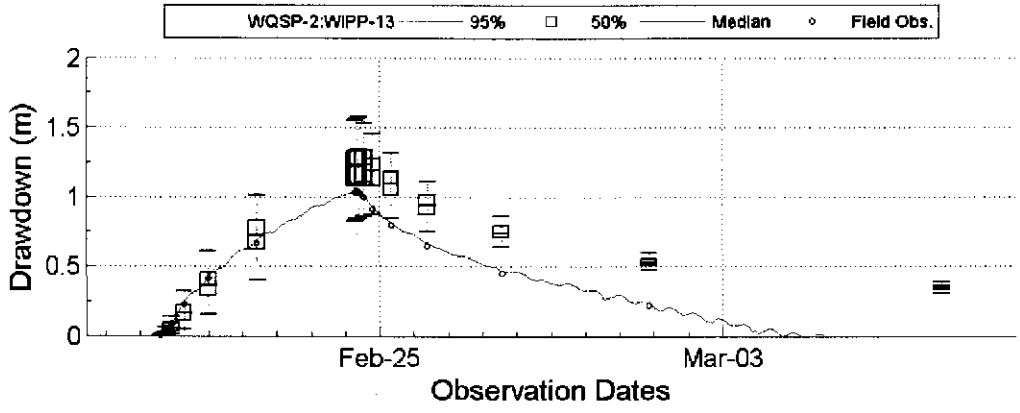


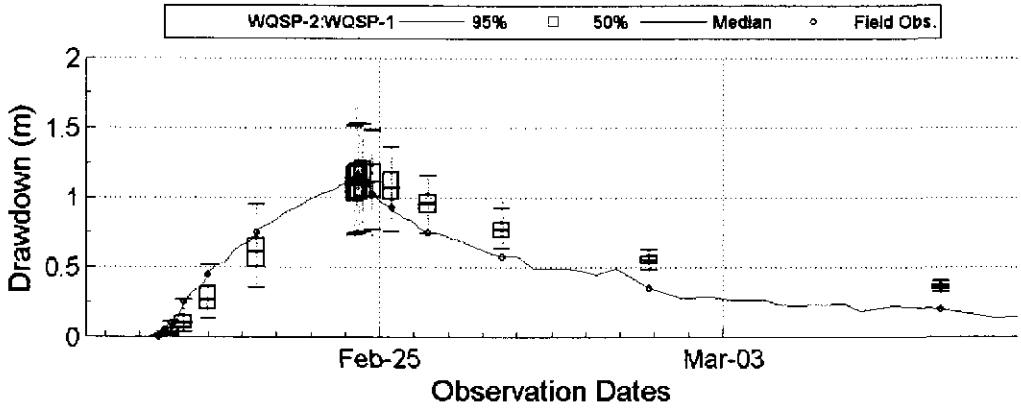


D.7 Modeled Responses to the WQSP-2 Pumping Test

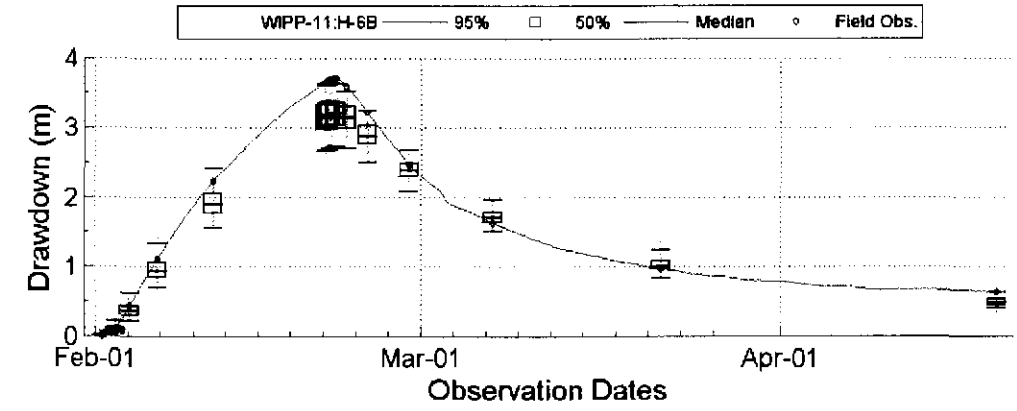


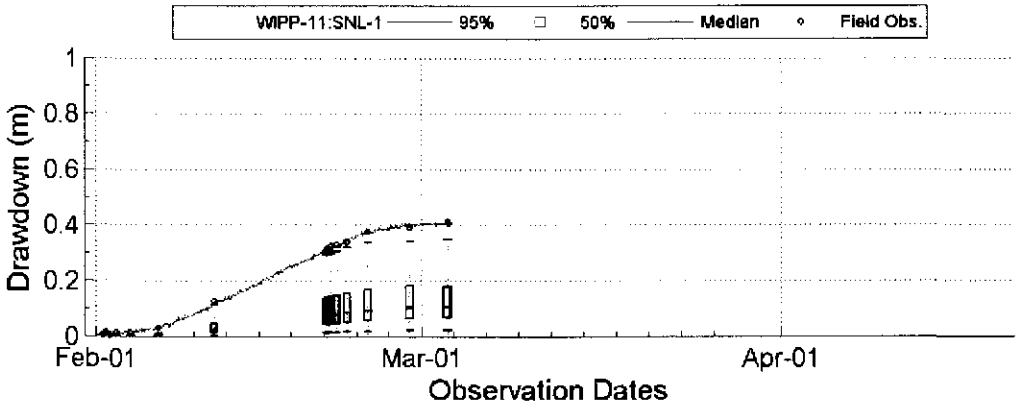


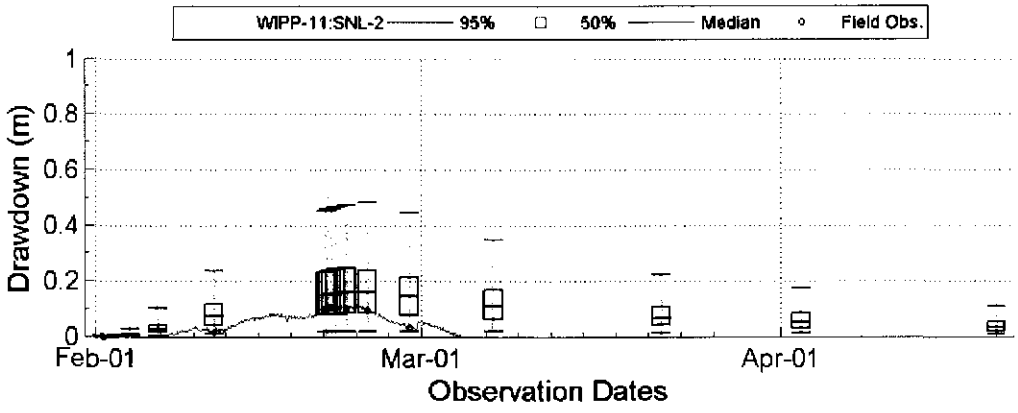


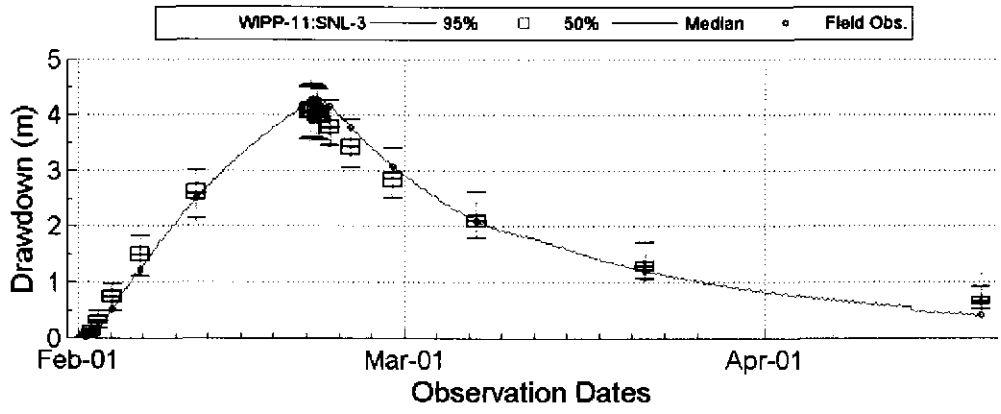


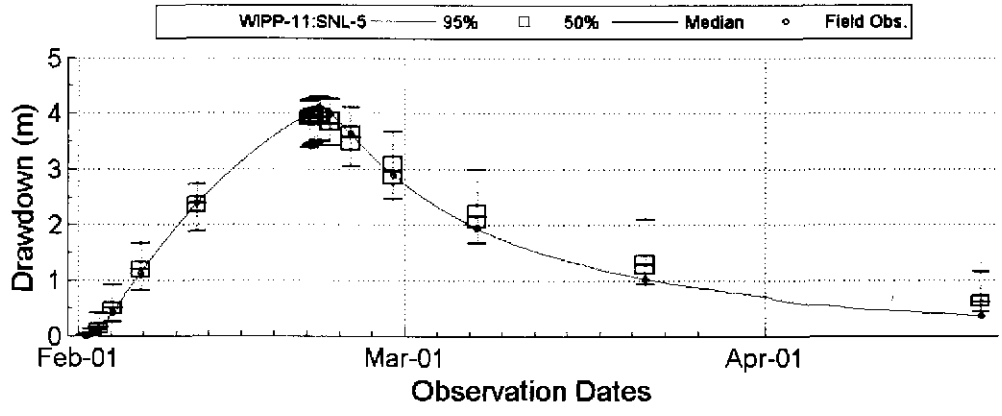
D.8 Modeled Responses to the WIPP-11 Pumping Test

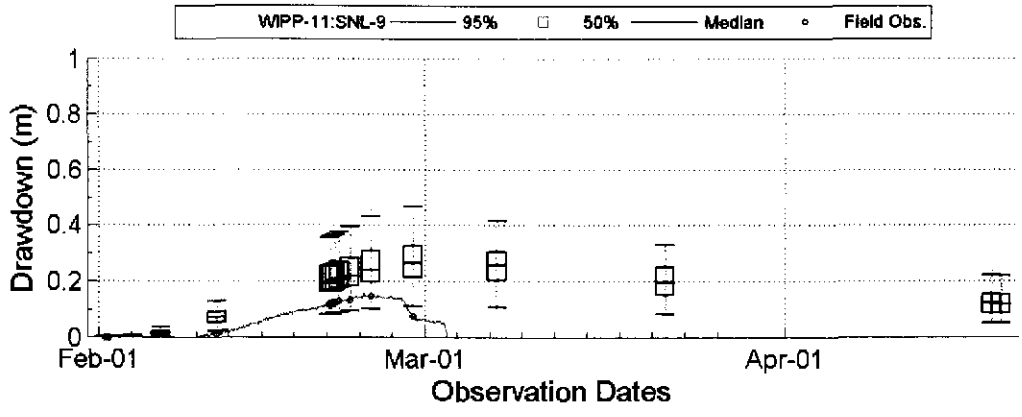


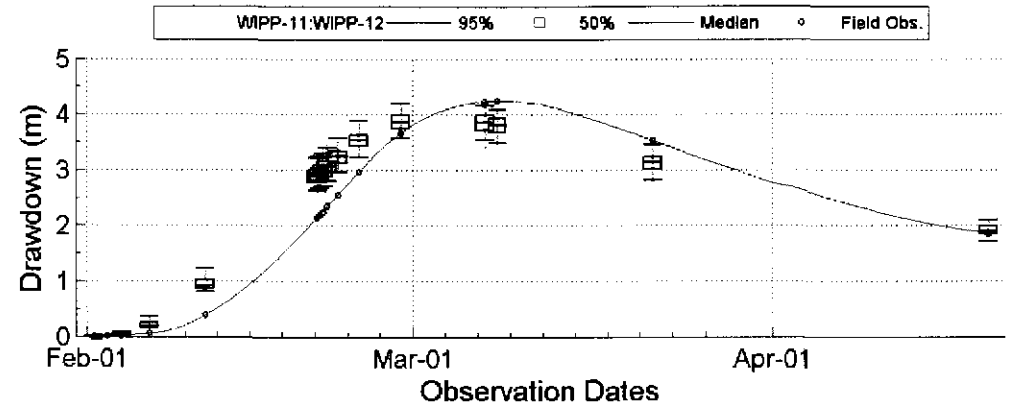


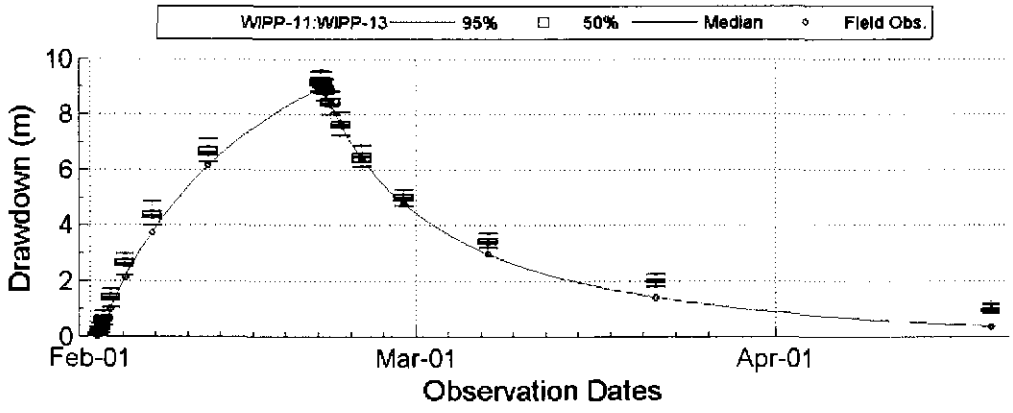


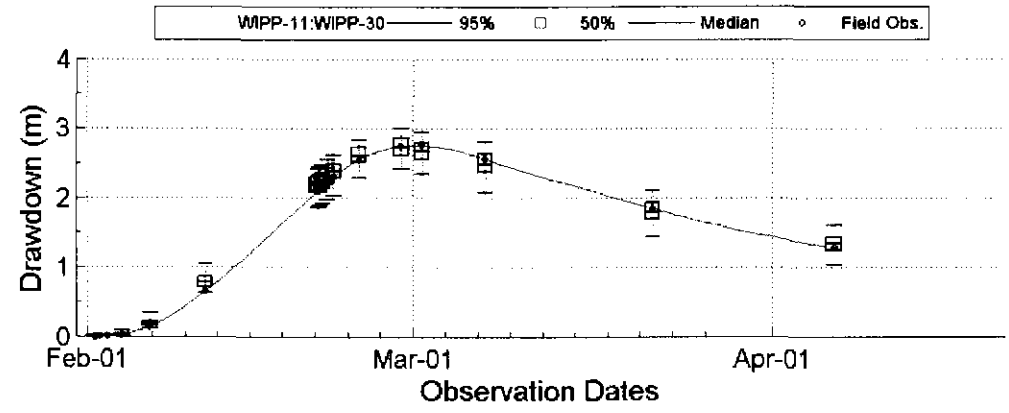


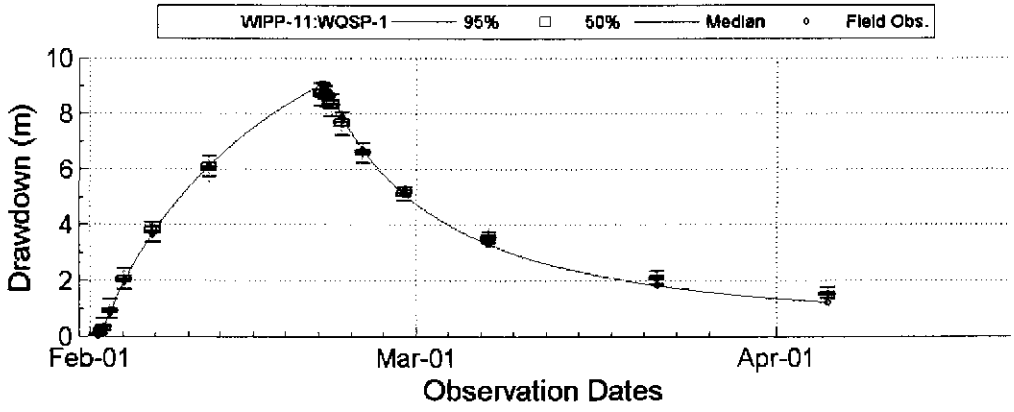


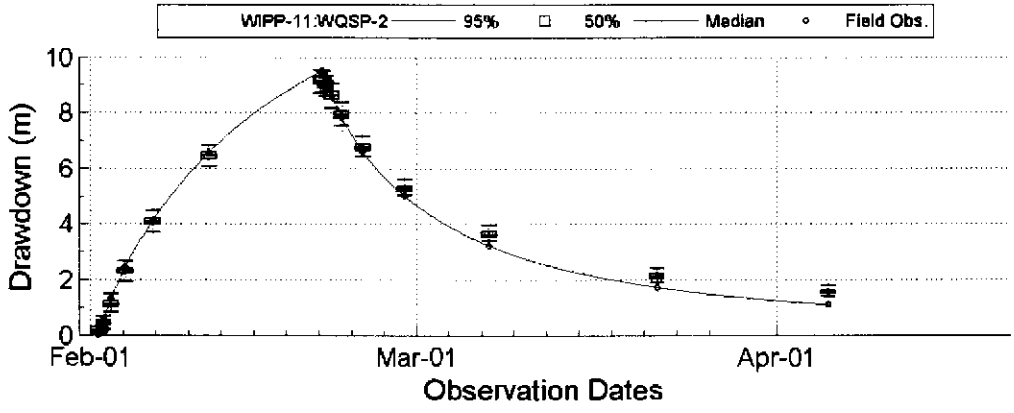


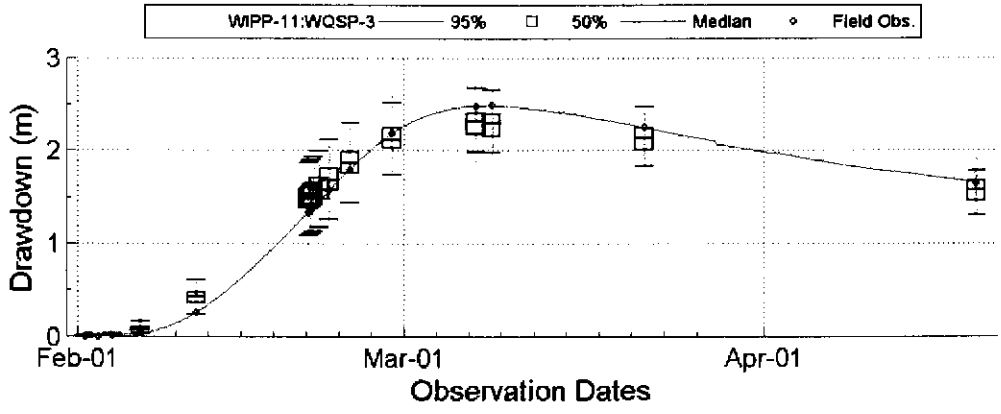




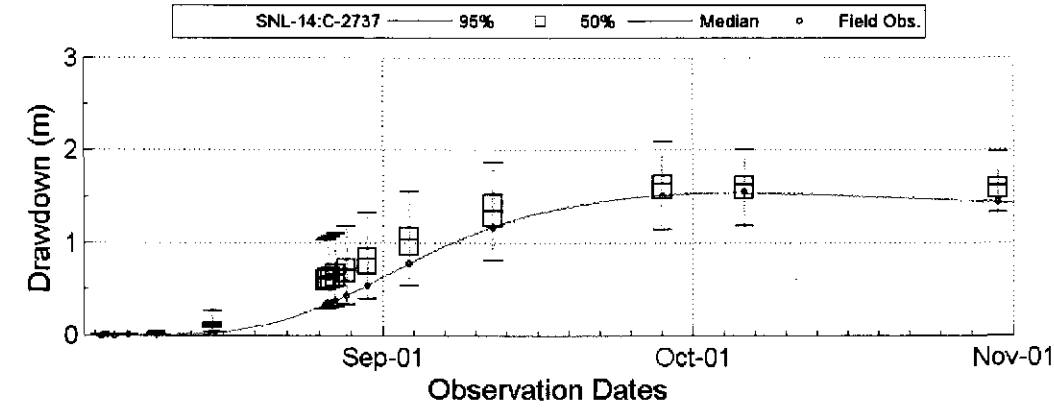


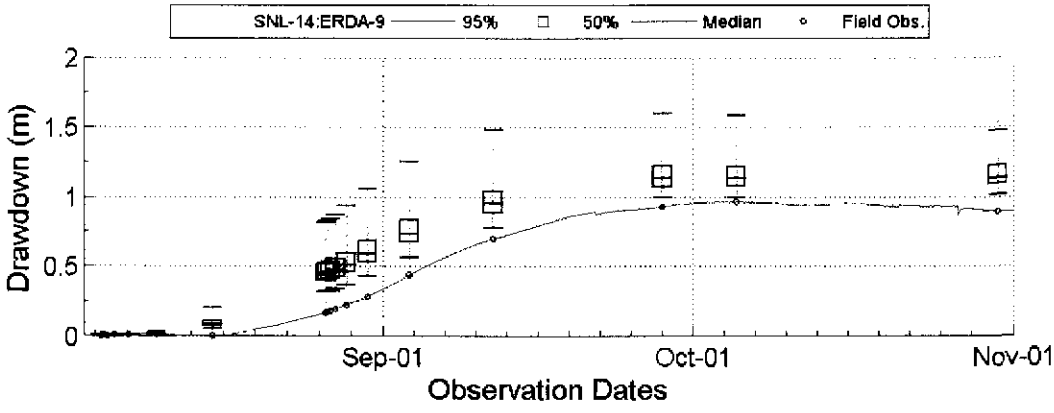


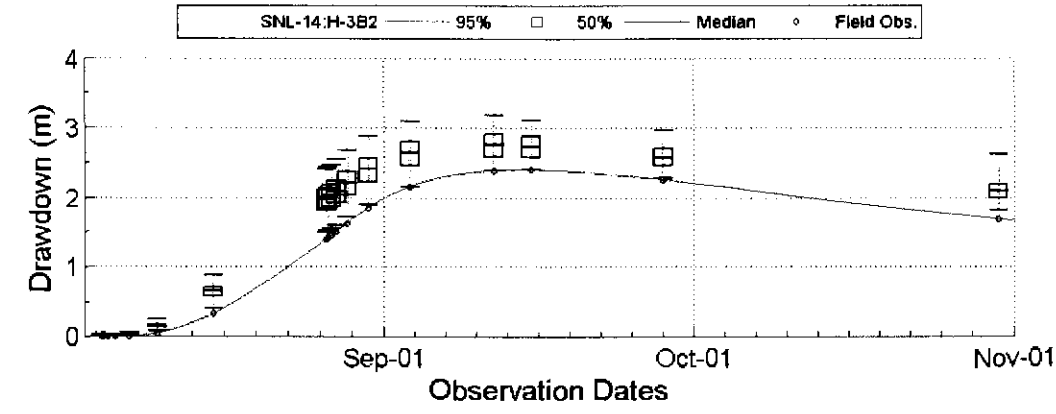


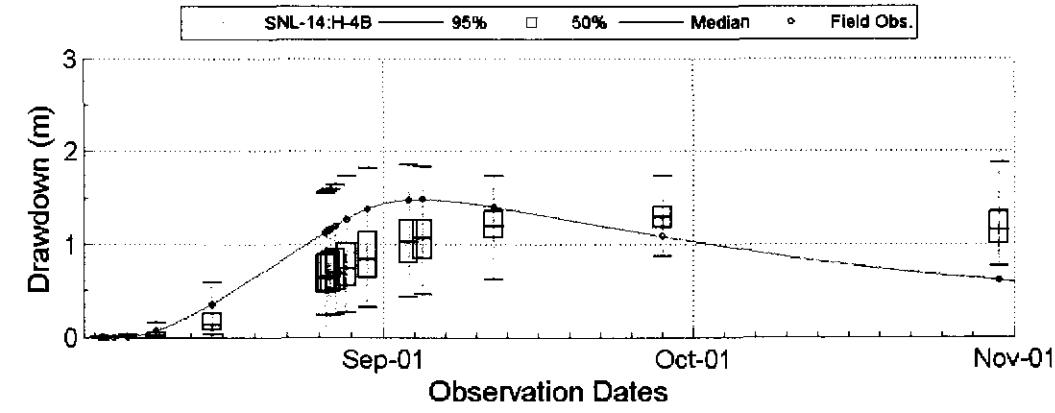


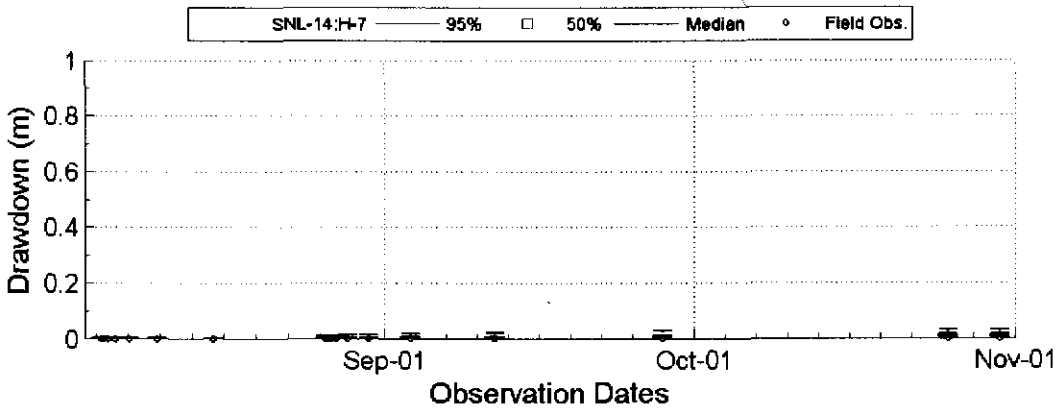
D.9 Modeled Responses to the SNL-14 Pumping Test

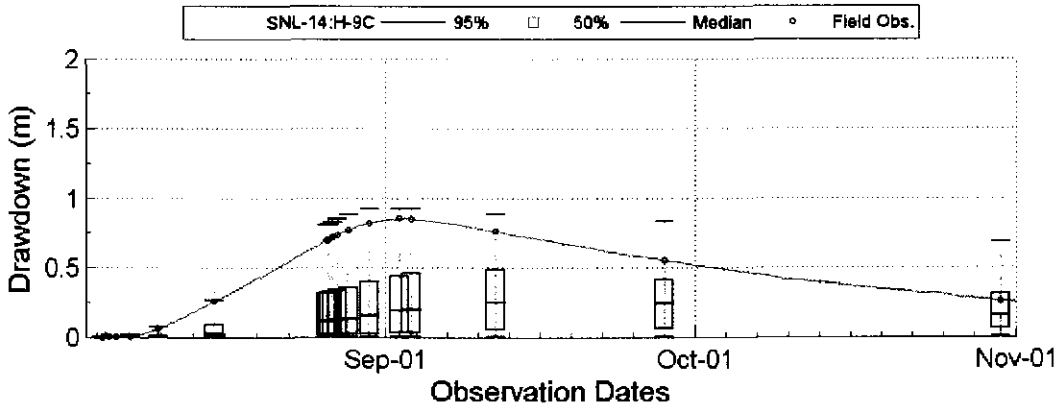


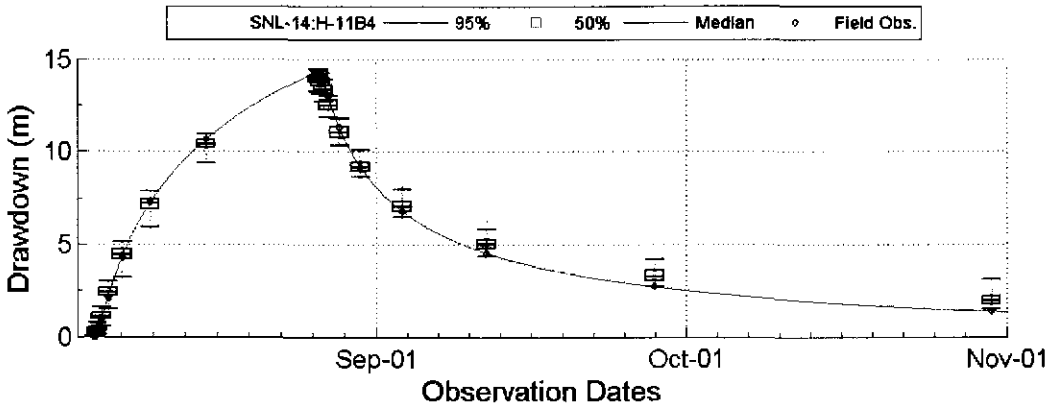


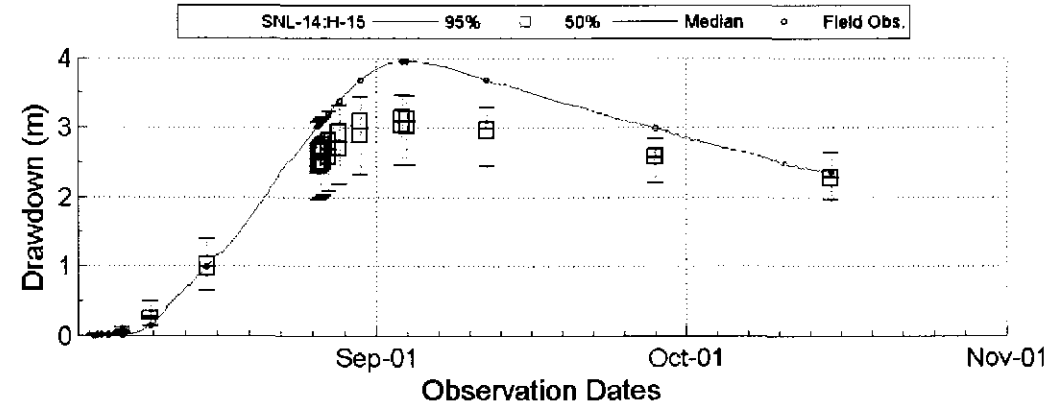


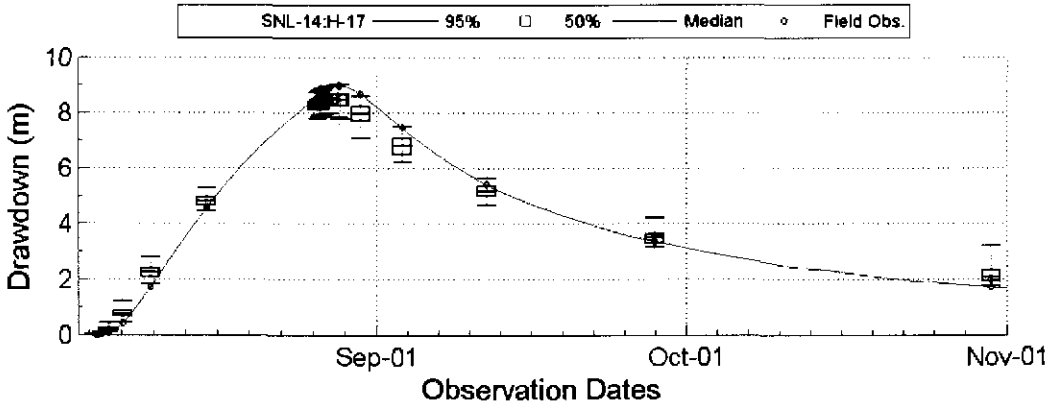


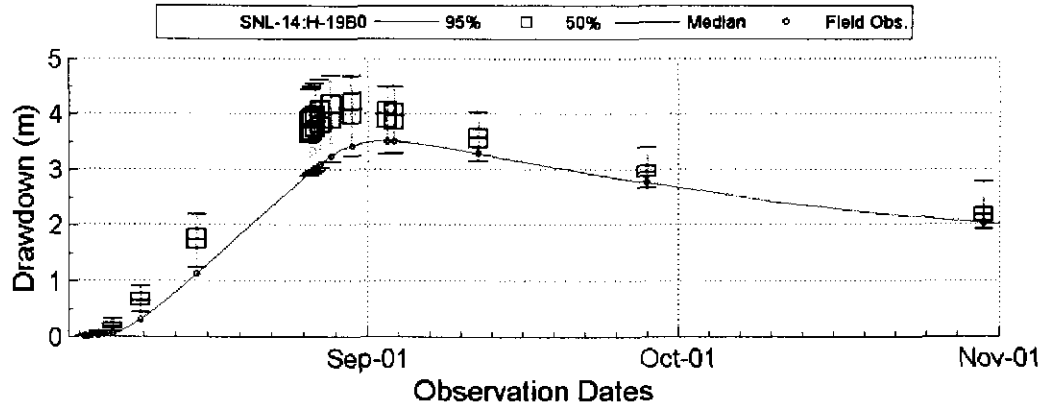


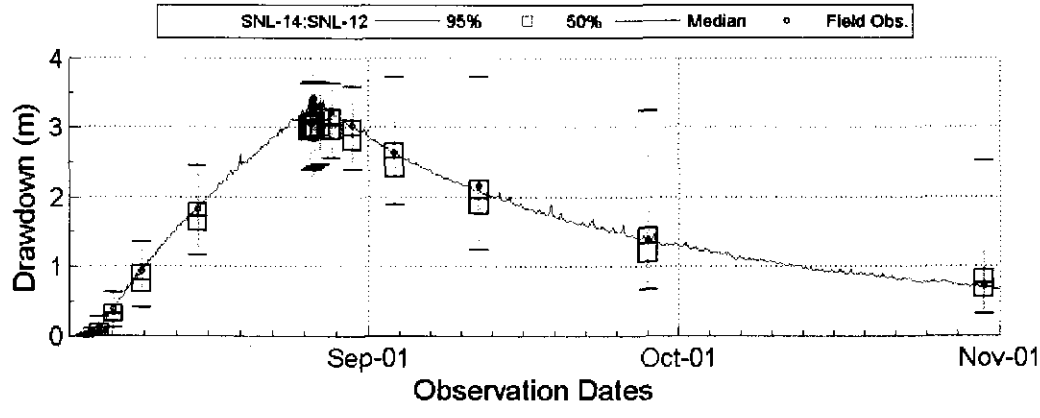


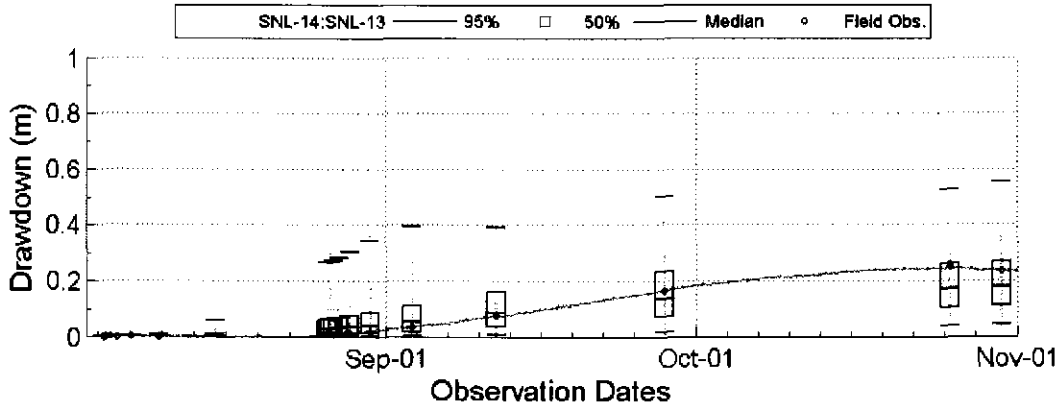












[THIS PAGE INTENTIONALLY LEFT BLANK]

Appendix E: Tabulated Calibration Results

The following is a tabulated list of the results from all 200 calibrated fields in order by base field identifier. Each row contains: the field ID, the field set (A: Recalibrated; B: Calibrated; C: 2x-Recalibrated), whether the field is in the final selected fields, the average steady-state head error in **centimeters**, the average transient pumping test error in **centimeters**, the 7.75-m thickness advective travel time to the land withdrawal boundary and the 4-m thickness advective travel time.

Field ID	Calibration Set	Steady State Average Error (cm)	Pumping Response Average Error (cm)	Selected Fields	Travel 7.75-m (y)	Travel 4-m (y)
r000	A	81.07	18.23	-	14831	7655
r001	A	65.59	15.78	Final	7152	3691
r002	A	64.39	15.24	Final	9094	4694
r003	A	91.04	17.75	-	23669	12217
r004	B	62.71	15.97	Final	28905	14919
r005	A	64.14	17.35	-	21058	10868
r006	A	55.91	13.83	Final	5447	2812
r007	C	64.10	14.37	Final	16916	8731
r008	A	100.52	17.26	-	12683	6546
r009	A	64.98	15.24	Final	24687	12742
r010	A	55.37	13.84	Final	18239	9414
r011	A	53.59	17.05	-	21555	11125
r012	A	54.32	15.91	Final	30523	15754
r013	C	68.79	15.19	Final	13983	7217
r014	A	93.14	19.11	-	6317	3260
r015	A	69.42	17.67	-	8980	4635
r016	A	94.64	18.63	-	8637	4458
r017	C	67.84	14.22	Final	14099	7277
r018	B	98.77	17.86	-	35321	18230
r019	A	68.32	17.27	-	8323	4296
r021	B	66.73	17.37	-	5678	2931
r022	B	112.34	20.11	-	11227	5794
r023	B	106.54	22.99	-	10413	5374
r024	B	69.37	15.94	Final	9061	4677
r026	B	66.34	19.51	-	36402	18788
r027	B	62.06	12.45	Final	10763	5555
r028	B	50.46	14.41	Final	33094	17081
r029	B	57.31	15.13	Final	37387	19296
r031	A	72.28	17.65	-	11203	5782

Field ID	Calibration Set	Steady State Average Error (cm)	Pumping Response Average Error (cm)	Selected Fields	Travel 7.75-m (y)	Travel 4-m (y)
r032	A	64.65	14.44	Final	9873	5096
r033	A	86.34	18.08	-	44912	23180
r034	A	67.13	15.12	Final	11158	5759
r035	A	90.27	16.56	-	39496	20385
r036	A	66.89	17.69	-	14549	7509
r037	A	61.62	15.90	Final	38390	19814
r038	A	48.93	14.77	Final	15409	7953
r039	A	67.33	18.50	-	43732	22571
r040	A	59.60	15.98	Final	71972	37147
r041	A	67.33	15.66	Final	15738	8123
r042	A	66.77	17.49	-	9536	4922
r043	B	86.15	16.28	-	14809	7643
r044	A	70.67	16.93	-	13992	7222
r045	A	58.45	16.15	Final	7976	4117
r046	A	87.22	20.76	-	12981	6700
r047	B	63.89	16.74	-	20057	10352
r048	A	81.40	22.25	-	8745	4514
r049	A	60.42	16.85	-	7128	3679
r050	A	71.39	18.79	-	11186	5774
r051	A	59.45	13.79	Final	23764	12265
r052	A	57.15	14.93	Final	39215	20240
r053	A	62.73	16.36	Final	15352	7924
r054	A	52.68	13.49	Final	18407	9500
r055	A	64.51	15.73	Final	30067	15518
r056	A	71.42	14.05	-	6565	3388
r057	A	64.40	17.03	-	12034	6211
r058	A	63.87	15.44	Final	79964	41272
r059	A	52.28	16.28	Final	18194	9390
r060	A	54.55	15.73	Final	18765	9685
r061	A	65.92	14.75	Final	9877	5098
r062	A	78.65	18.43	-	8524	4399
r063	B	115.33	22.05	-	12717	6564
r064	A	62.69	15.85	Final	14416	7441
r065	A	69.87	15.94	-	16160	8340
r066	A	129.90	23.15	-	7905	4080
r067	A	96.78	19.59	-	8327	4298

Field ID	Calibration Set	Steady State Average Error (cm)	Pumping Response Average Error (cm)	Selected Fields	Travel 7.75-m (y)	Travel 4-m (y)
r068	C	69.72	16.22	-	10202	5266
r069	A	77.96	18.57	-	14968	7725
r070	A	63.30	14.10	Final	14434	7450
r071	B	78.50	15.55	-	11499	5935
r072	B	102.28	17.15	-	12513	6458
r073	B	59.90	14.26	Final	33710	17399
r074	B	65.43	14.46	Final	18055	9319
r075	B	73.55	17.16	-	17109	8831
r076	B	55.28	13.83	Final	13516	6976
r077	B	72.50	15.86	-	9067	4680
r078	B	56.21	14.66	Final	33382	17230
r079	B	60.07	16.38	-	14636	7554
r080	B	65.43	16.86	-	8617	4447
r081	B	88.69	18.28	-	13205	6816
r082	B	60.58	13.29	Final	21281	10984
r083	B	54.69	13.28	Final	19296	9959
r084	B	61.87	15.33	Final	9196	4746
r085	B	76.15	17.30	-	9536	4922
r086	B	71.04	15.80	-	39643	20461
r087	A	71.83	18.37	-	9241	4769
r088	B	92.99	20.04	-	11647	6011
r089	B	82.31	18.18	-	20750	10710
r090	B	53.99	14.52	Final	9013	4652
r091	B	72.00	17.24	-	12495	6449
r092	A	60.90	14.01	Final	19718	10177
r093	B	94.46	17.29	-	21402	11046
r094	B	76.58	15.67	-	27838	14368
r095	A	66.49	15.96	Final	7194	3713
r096	B	65.15	16.59	-	19323	9973
r097	B	55.24	14.48	Final	17066	8808
r098	B	54.02	13.03	Final	24384	12585
r099	B	72.28	15.81	-	31940	16485
r100	A	80.08	22.93	-	8451	4362
r101	B	83.30	17.08	-	8271	4269
r102	B	58.80	13.89	Final	38079	19654
r103	B	111.22	18.22	-	15901	8207

Field ID	Calibration Set	Steady State Average Error (cm)	Pumping Response Average Error (cm)	Selected Fields	Travel 7.75-m (y)	Travel 4-m (y)
r104	B	55.40	15.35	Final	21870	11288
r105	B	67.51	18.16	-	17013	8781
r106	A	70.05	12.92	-	10819	5584
r107	B	76.60	14.95	-	73697	38037
r108	B	83.67	16.17	-	18871	9740
r109	B	56.88	17.39	-	42665	22021
r117	A	73.49	19.06	-	10945	5649
r118	B	90.83	15.85	-	6052	3124
r119	B	82.24	18.52	-	12926	6671
r120	B	81.91	17.72	-	3116	1608
r133	A	93.19	21.17	-	13843	7145
r137	C	61.74	14.17	Final	9853	5085
r140	A	87.10	17.48	-	12147	6269
r141	A	194.99	27.87	-	11202	5782
r142	A	68.00	15.35	Final	90229	46570
r149	A	99.18	18.37	-	10221	5275
r151	A	139.79	34.21	-	6734	3476
r191	A	64.48	15.47	Final	9810	5063
r203	A	66.17	15.39	Final	5965	3079
r207	C	60.36	13.93	Final	10002	5162
r228	A	71.26	15.24	-	10776	5562
r245	A	81.15	16.63	-	7834	4043
r256	A	45.59	15.88	Final	25888	13362
r257	A	72.38	17.78	-	37561	19387
r260	A	69.33	14.95	Final	10978	5666
r272	A	74.42	19.75	-	17708	9140
r273	A	61.08	15.71	Final	19453	10040
r276	A	61.38	16.07	Final	16163	8342
r279	A	61.02	14.69	Final	12900	6658
r298	A	49.15	15.30	Final	35071	18101
r312	A	119.32	21.46	-	14678	7576
r326	C	72.79	16.63	-	9145	4720
r327	C	64.76	15.81	Final	7905	4080
r328	A	57.66	15.41	Final	7373	3806
r332	A	59.05	17.22	-	8411	4341
r348	A	74.92	16.37	-	8141	4202

Field ID	Calibration Set	Steady State Average Error (cm)	Pumping Response Average Error (cm)	Selected Fields	Travel 7.75-m (y)	Travel 4-m (y)
r353	A	80.65	18.60	-	19977	10311
r359	A	78.44	15.26	-	17541	9054
r361	A	61.97	14.33	Final	13794	7119
r384	A	72.09	15.85	-	24620	12707
r404	A	67.35	16.63	-	9213	4755
r431	A	55.90	15.13	Final	11891	6137
r440	A	60.97	15.46	Final	4982	2571
r452	A	70.63	16.06	-	11429	5899
r465	C	65.48	14.62	Final	8153	4208
r473	A	66.16	16.96	-	5970	3081
r486	A	51.75	13.13	Final	10277	5304
r489	C	69.60	14.85	Final	10755	5551
r506	A	60.81	13.51	Final	9168	4732
r508	A	59.92	14.49	Final	22407	11565
r511	A	66.10	14.28	Final	14416	7441
r515	A	65.92	15.19	Final	13721	7082
r522	A	58.64	14.88	Final	37305	19254
r527	A	83.89	17.85	-	18442	9518
r568	A	62.93	14.70	Final	50379	26002
r569	A	66.61	16.47	-	12118	6254
r571	A	62.84	13.98	Final	9323	4812
r593	A	74.59	17.43	-	38887	20071
r598	A	63.41	18.39	-	19294	9958
r606	A	80.93	16.15	-	6162	3180
r631	A	57.47	13.46	Final	14132	7294
r634	A	64.53	14.19	Final	31995	16514
r640	A	54.72	14.81	Final	12785	6599
r643	C	65.55	17.35	-	15360	7928
r647	C	70.01	15.45	-	4619	2384
r650	A	82.98	18.57	-	14838	7659
r652	A	63.41	15.26	Final	14157	7307
r655	A	58.67	14.38	Final	21569	11132
r657	A	54.96	12.86	Final	13498	6967
r664	A	52.72	14.34	Final	51434	26546
r669	A	64.11	13.72	Final	10216	5273
r694	A	60.86	15.06	Final	32963	17013

Field ID	Calibration Set	Steady State Average Error (cm)	Pumping Response Average Error (cm)	Selected Fields	Travel 7.75-m (y)	Travel 4-m (y)
r707	A	58.08	15.61	Final	12210	6302
r727	A	69.05	15.95	Final	20053	10350
r731	A	75.01	16.95	-	48421	24992
r752	A	65.92	14.23	Final	15672	8089
r791	A	48.93	15.86	Final	15888	8200
r806	A	68.81	16.17	Final	9366	4834
r808	A	55.40	12.95	Final	7511	3877
r809	A	58.19	14.45	Final	12152	6272
r814	A	58.28	15.60	Final	13481	6958
r823	A	57.17	15.55	Final	12720	6565
r831	A	70.77	15.92	-	10322	5328
r848	A	62.25	16.78	-	7252	3743
r861	A	63.12	15.68	Final	14957	7720
r865	A	64.50	18.10	-	16042	8280
r880	C	66.46	16.71	-	13543	6990
r883	A	62.42	13.24	Final	11062	5709
r902	A	48.49	14.82	Final	16580	8557
r908	A	64.17	18.17	-	17324	8942
r910	A	50.16	15.10	Final	143204	73912
r921	A	57.54	15.00	Final	8427	4350
r922	A	55.64	14.38	Final	14006	7229
r937	A	68.90	17.53	-	44608	23023
r940	C	65.93	15.85	Final	5818	3003
r981	C	57.84	15.13	Final	34084	17592
r982	A	53.02	15.34	Final	77947	40231
r984	A	50.66	13.90	Final	29312	15129
r987	A	67.42	17.44	-	7733	3991

Appendix F: Files Used – Descriptions and Locations

Table F-1. Static input files located in the Inputs module of the CVS repository for this task.

File under CVS://Tfields::Inputs	Brief Description
./config	Directory containing configuration and input files
./config/dtrkmf_domain.inp	DTRKMF Input file for particle tracks extending to the
./config/dtrkmf_wipplwb.inp	DTRKMF Input file for particle tracks to the WIPP WLB
./config/fac2real_points_A.in	FAC2REAL input control file for anisotropy
./config/fac2real_points_R.in	FAC2REAL input control file for recharge
./config/fac2real_points_S.in	FAC2REAL input control file for storativity
./config/fac2real_points_T.in	FAC2REAL input control file for transmissivity
./config/mod2obs_h11.in	MOD2OBS input control file for the H-11 pumping test
./config/mod2obs_h19.in	MOD2OBS input control file for the H-19 pumping test
./config/mod2obs_h3.in	MOD2OBS input control file for the H-3 pumping test
./config/mod2obs_head.in	MOD2OBS input control file for the steady-state test
./config/mod2obs_p14.in	MOD2OBS input control file for the P-14 pumping test
./config/mod2obs_snl14.in	MOD2OBS input control file for the SNL-14 pumping test
./config/mod2obs_wipp11.in	MOD2OBS input control file for the WIPP-11 pumping test
./config/mod2obs_wipp13.in	MOD2OBS input control file for the WIPP-13 pumping test
./config/mod2obs_wqsp1.in	MOD2OBS input control file for the WQSP-1 pumping test
./config/mod2obs_wqsp2.in	MOD2OBS input control file for the WQSP-2 pumping test
./config/obs2real_h11.in	OBS2REAL input control file for the H-11 pumping test
./config/obs2real_h19.in	OBS2REAL input control file for the H-19 pumping test
./config/obs2real_h3.in	OBS2REAL input control file for the H-3 pumping test
./config/obs2real_head.in	OBS2REAL input control file for the steady-state test
./config/obs2real_p14.in	OBS2REAL input control file for the P-14 pumping test
./config/obs2real_snl14.in	OBS2REAL input control file for the SNL-14 pumping test
./config/obs2real_wipp11.in	OBS2REAL input control file for the WIPP-11 pumping test
./config/obs2real_wipp13.in	OBS2REAL input control file for the WIPP-13 pumping test
./config/obs2real_wqsp1.in	OBS2REAL input control file for the WQSP-1 pumping test
./config/obs2real_wqsp2.in	OBS2REAL input control file for the WQSP-2 pumping test
./config/ppk2fac_points_A.in	PPK2FAC input control file for anisotropy
./config/ppk2fac_points_R.in	PPK2FAC input control file for recharge
./config/ppk2fac_points_S.in	PPK2FAC input control file for storativity
./config/ppk2fac_points_T.in	PPK2FAC input control file for transmissivity
./config/settings.fig	Configuration file for PEST programs
./config/spec_connect_A.lst	INIT2PEST well connections input data for anisotropy
./config/spec_connect_R.lst	INIT2PEST well connections input data for recharge
./config/spec_connect_S.lst	INIT2PEST well connections input data for storativity

File under CVS://Tfields::Inputs	Brief Description
config/spec_connect_T.lst	INIT2PEST well connections input data for transmissivity
config/spec_domain.spc	Grid specification file for PEST programs
config/spec_info.lst	Information regarding recharge and travel times
config/spec_locs.crd	Well names and coordinates
config/spec_observations.in	INIT2PEST input file - see Appendix G
config/spec_parameters.lst	List of parameters (T, S, R, and A)
config/spec_points_A.in	INIT2PEST input file - see Appendix G
config/spec_points_R.in	INIT2PEST input file - see Appendix G
config/spec_points_S.in	INIT2PEST input file - see Appendix G
config/spec_points_T.in	INIT2PEST input file - see Appendix G
config/spec_pumping.dat	Information regarding pumping dates and times
config/spec_recharge.crd	Recharge locations and coordinates
config/spec_setup.in	INIT2PEST input file - see Appendix G
config/spec_slavefiles.lst	List of files to copy to each slave node
config/spec_steady.lst	List of steady-state test IDs
config/spec_testdates.dat	Information regarding pumping dates and times
config/spec_transient.lst	List of transient test IDs
config/spec_variogram.str	Variogram structure file
config/spec_weights.lst	Observation Weights
config/spec_wells.crd	Well names and coordinates
data	Directory containing hard data
data/elev_bot.mod	Elevation of bottom of the Culebra
data/elev_culebra.geo	Elevation of the center of the Culebra
data/elev_culebra.mod	Elevation of the center of the Culebra
data/elev_overburden.geo	Overburden thickness map
data/elev_overburden.mod	Overburden thickness map
data/elev_surface.mod	Ground surface elevation map
data/elev_top.mod	Elevation of the top of the Culebra
data/fixed_points_A.ppt	Fixed pilot point locations and values for anisotropy
data/fixed_points_R.ppt	Fixed pilot point locations and values for recharge
data/fixed_points_S.ppt	Fixed pilot point locations and values for storativity
data/fixed_points_T.ppt	Fixed pilot point locations and values for transmissivity
data/init_bnds.inf	Initial head field boundary conditions
data/init_head.mod	Initial head field
data/meas_h11.smp	Pumping test observations
data/meas_h19.smp	Pumping test observations
data/meas_h3.smp	Pumping test observations
data/meas_head.smp	Steady state head observations
data/meas_p14.smp	Pumping test observations
data/meas_sn114.smp	Pumping test observations

File under CVS://Tfields::Inputs	Brief Description
data/meas_wipp11.smp	Pumping test observations
data/meas_wipp13.smp	Pumping test observations
data/meas_wqsp1.smp	Pumping test observations
data/meas_wqsp2.smp	Pumping test observations
data/zone_central.geo	Central zone extent definition file
data/zone_confined.geo	Confined zone extent definition file
data/zone_dissolution.geo	Dissolution zone extent definition file
data/zone_dissolution.inf	Dissolution zone extent definition file
data/zone_domain.geo	Domain zone definition file
data/zone_halite.geo	Halite margins definition file
data/zone_halite.inf	Halite margins definition file
data/zone_halitemargins.geo	Halite margins definition file
data/zone_noflow.geo	No-flow boundary definition file
data/zone_recharge.geo	Recharge zone definition file
data/zone_transition.geo	Transition (confined-unconfined) definition file
data/zone_unconfined.geo	Unconfined zone definition file
modflow	MODFLOW control files out
modflow/mf2k_culebra.ba6	MODFLOW basic package control file
modflow/mf2k_culebra.dis	MODFLOW discretization package control file
modflow/mf2k_culebra.lmg	MODFLOW AMG solver control file
modflow/mf2k_culebra.lpf	MODFLOW layer package flow control file
modflow/mf2k_culebra.oc	MODFLOW output control file
modflow/mf2k_culebra.pcg	MODFLOW PCG solver control file
modflow/mf2k_culebra.rch	MODFLOW recharge package control file
modflow/mf2k_culebra.wel	MODFLOW well package control file
modflow/mf2k_h11.dis	Pumping test time discretization file
modflow/mf2k_h11.nam	Pumping test control file
modflow/mf2k_h11.oc	Pumping test output control file
modflow/mf2k_h11.pcg.nam	Pumping test backup solver control file
modflow/mf2k_h11.rch	Pumping test recharge control file
modflow/mf2k_h11.wel	Pumping test well discharge control file
modflow/mf2k_h19.dis	Pumping test time discretization file
modflow/mf2k_h19.nam	Pumping test control file
modflow/mf2k_h19.oc	Pumping test output control file
modflow/mf2k_h19.pcg.nam	Pumping test backup solver control file
modflow/mf2k_h19.rch	Pumping test recharge control file
modflow/mf2k_h19.wel	Pumping test well discharge control file
modflow/mf2k_h3.dis	Pumping test time discretization file
modflow/mf2k_h3.nam	Pumping test control file
modflow/mf2k_h3.oc	Pumping test output control file

File under CVS://Tfields::Inputs	Brief Description
modflow/mf2k_h3.pcg.nam	Pumping test backup solver control file
modflow/mf2k_h3.rch	Pumping test recharge control file
modflow/mf2k_h3.wel	Pumping test well discharge control file
modflow/mf2k_head.ba6	Steady state head basic control file
modflow/mf2k_head.dis	Steady state head discretization control file
modflow/mf2k_head.nam	Steady state head control file
modflow/mf2k_head.oc	Steady state head output control file
modflow/mf2k_head.rch	Steady state head recharge control file
modflow/mf2k_p14.dis	Pumping test time discretization file
modflow/mf2k_p14.nam	Pumping test control file
modflow/mf2k_p14.oc	Pumping test output control file
modflow/mf2k_p14.pcg.nam	Pumping test backup solver control file
modflow/mf2k_p14.rch	Pumping test recharge control file
modflow/mf2k_p14.wel	Pumping test well discharge control file
modflow/mf2k_snl14.dis	Pumping test time discretization file
modflow/mf2k_snl14.nam	Pumping test control file
modflow/mf2k_snl14.oc	Pumping test output control file
modflow/mf2k_snl14.pcg.nam	Pumping test backup solver control file
modflow/mf2k_snl14.rch	Pumping test recharge control file
modflow/mf2k_snl14.wel	Pumping test well discharge control file
modflow/mf2k_wipp11.dis	Pumping test time discretization file
modflow/mf2k_wipp11.nam	Pumping test control file
modflow/mf2k_wipp11.oc	Pumping test output control file
modflow/mf2k_wipp11.pcg.nam	Pumping test backup solver control file
modflow/mf2k_wipp11.rch	Pumping test recharge control file
modflow/mf2k_wipp11.wel	Pumping test well discharge control file
modflow/mf2k_wipp13.dis	Pumping test time discretization file
modflow/mf2k_wipp13.nam	Pumping test control file
modflow/mf2k_wipp13.oc	Pumping test output control file
modflow/mf2k_wipp13.pcg.nam	Pumping test backup solver control file
modflow/mf2k_wipp13.rch	Pumping test recharge control file
modflow/mf2k_wipp13.wel	Pumping test well discharge control file
modflow/mf2k_wqsp1.dis	Pumping test time discretization file
modflow/mf2k_wqsp1.nam	Pumping test control file
modflow/mf2k_wqsp1.oc	Pumping test output control file
modflow/mf2k_wqsp1.pcg.nam	Pumping test backup solver control file
modflow/mf2k_wqsp1.rch	Pumping test recharge control file
modflow/mf2k_wqsp1.wel	Pumping test well discharge control file
modflow/mf2k_wqsp2.dis	Pumping test time discretization file
modflow/mf2k_wqsp2.nam	Pumping test control file

File under CVS://Tfields::Inputs	Brief Description
modflow/mf2k_wqsp2.oc	Pumping test output control file
modflow/mf2k_wqsp2.pcg.nam	Pumping test backup solver control file
modflow/mf2k_wqsp2.rch	Pumping test recharge control file
modflow/mf2k_wqsp2.wel	Pumping test well discharge control file
scripts	Directory containing analyst scripts -- See Appendix G
scripts/fix_snl8.sh	Script to recalibrate locally to SNL-8
scripts/init2pest.py	Setup and control creation program
scripts/model.sh	Forward model
scripts/obs2real.py	OBS2REAL program
scripts/pestutil.py	PESTUTIL library
scripts/real2mod.py	REAL2MOD program
scripts/run_dtrkmf.sh	DTRKMF analysis script
scripts/runpest.sh	Run the PEST calibration process
scripts/setup.sh	Setup the calibration
scripts/setup10.sh	Setup the calibration for limited iterations
scripts/template_pmaster.sh	Master node control script
scripts/template_pmaster10.sh	Master node control script (10 iterations)
scripts/template_pslave.sh	Slave node control script

Table F-2. Files created by SETUP and INIT2PEST

File created during setup	Brief Description [*used by*]
modeled_basefield.map	Copied from r???coord.map checked out via RunReadScript [*INIT2PEST*]
modeled_fieldID.tex	Text store of the field being calibrated
modeled_head.ins	Instruction file for steady-state head observations [*PEST*]
modeled_h3.ins	Instruction file for H-3 pumping test observations [*PEST*]
modeled_wipp13.ins	Instruction file for WIPP-13 pumping test observations [*PEST*]
modeled_h11.ins	Instruction file for H-11 pumping test observations [*PEST*]
modeled_p14.ins	Instruction file for P-14 pumping test observations [*PEST*]
modeled_h19.ins	Instruction file for H-19 pumping test observations [*PEST*]
modeled_wqsp1.ins	Instruction file for WQSP-1 pumping test observations [*PEST*]
modeled_wqsp2.ins	Instruction file for WQSP-2 pumping test observations [*PEST*]
modeled_wipp11.ins	Instruction file for WIPP-11 pumping test observations [*PEST*]
modeled_snl14.ins	Instruction file for SNL-14 pumping test observations [*PEST*]
modeled_zones_points_T.inf	Zone file [*PPK2FAC*]
modeled_init_points_T.mod	Initial field values [*REAL2MOD*]
modeled_points_T.tpl	Template file for pilot points [*PEST*]
modeled_points_T.dat	Initial pilot point values [*PPK2FAC, FAC2REAL*] Written new for each forward call by PEST
real2mod_points_T.tpl	Template file for shift/log factor [*PEST*]
real2mod_points_T.in	Input file for shift/log factors [*REAL2MOD*]
ppk2fac_points_T.in	Input file for kriging factors creation [*PPK2FAC*]
modeled_init_points_T.tex	Output summary log
fac2real_points_T.in	Input file for pilot point kriging routine [*FAC2REAL*]
modeled_zones_points_S.inf	Zone file [*PPK2FAC*]
modeled_init_points_S.mod	Initial field values [*REAL2MOD*]
modeled_points_S.tpl	Template file for pilot points [*PEST*]
modeled_points_S.dat	Initial pilot point values [*PPK2FAC, FAC2REAL*]
real2mod_points_S.tpl	Template file for shift/log factor [*PEST*]
real2mod_points_S.in	Input file for shift/log factors [*REAL2MOD*]
ppk2fac_points_S.in	Input file for kriging factors creation [*PPK2FAC*]
modeled_init_points_S.tex	Output summary log
fac2real_points_S.in	Input file for pilot point kriging routine [*FAC2REAL*]
modeled_zones_points_A.inf	Zone file [*PPK2FAC*]
modeled_init_points_A.mod	Initial field values [*REAL2MOD*]

File created during setup	Brief Description [*used by*]
modeled_points_A.tpl	Template file for pilot points [*PEST*]
modeled_points_A.dat	Initial pilot point values [*PPK2FAC, FAC2REAL*]
real2mod_points_A.tpl	Template file for shift/log factor [*PEST*]
real2mod_points_A.in	Input file for shift/log factors [*REAL2MOD*]
ppk2fac_points_A.in	Input file for kriging factors creation [*PPK2FAC*]
modeled_init_points_A.tex	Output summary log
fac2real_points_A.in	Input file for pilot point kriging routine [*FAC2REAL*]
modeled_zones_points_R.inf	Zone file [*PPK2FAC*]
modeled_init_points_R.mod	Initial field values [*REAL2MOD*]
modeled_points_R.tpl	Template file for pilot points [*PEST*]
modeled_points_R.dat	Initial pilot point values [*PPK2FAC, FAC2REAL*]
real2mod_points_R.tpl	Template file for shift/log factor [*PEST*]
real2mod_points_R.in	Input file for shift/log factors [*REAL2MOD*]
ppk2fac_points_R.in	Input file for kriging factors creation [*PPK2FAC*]
modeled_init_points_R.tex	Output summary log
fac2real_points_R.in	Input file for pilot point kriging routine [*FAC2REAL*]
r???_init.pst	Initial configuration file for Jacobian generation [*PEST*]
r???_once.pst	Single forward-model configuration file [*PEST*]
r???_init.rmf	Parallel processing control file [*PEST*]
r???_svda.rmf	Parallel processing control file [*PEST*]
pest_slave.sh	Created from template_pslave.sh, submitted to queue on cluster
pest_master.sh	Created from template_pmaster.sh, submitted to queue on cluster

Table F-3. Files saved in CVS in Outputs/r??? and Outputs/Update*/r???

File saved	Created by	Used by	Brief description
modeled_A_field.mod	REAL2MOD	MODFLOW	Real-space anisotropy multiplier field
modeled_K_field.mod	REAL2MOD	MODFLOW	Real-space hydraulic conductivity field (T/thickness)
modeled_R_field.mod	REAL2MOD	MODFLOW	Real-space recharge field
modeled_S_field.mod	REAL2MOD	MODFLOW	Real-space specific storage field (S/thickness)
modeled_factors_points_*.bin	PPK2FAC	FAC2REAL	Kriging factors file
modeled_fieldID.tex	INIT2PEST	N/A	Field ID record
modeled_flow.bud	MODFLOW	DTRKMF	MODFLOW budget file
modeled_*.bin	MODFLOW	MOD2OBS	Drawdown/head fields for each MODFLOW test
modeled_*.ins	INIT2PEST	PEST	Instruction files for each pumping/steady state model
modeled_*.lst	MODFLOW	N/A	List file output from MODFLOW
modeled_*.out	OBS2REAL	PEST	Observations in single column format
modeled_*.smp	MOD2OBS	OBS2REAL	Point/time drawdown/head values
modeled_init_points_*.mod	INIT2PEST	REAL2MOD	Initial points used to fill in shifted initial values in non-interpolated areas in outputs from FAC2REAL
modeled_points_*.dat	PEST or PARCALC	PPK2FAC, FAC2REAL	Pilot points data files
modeled_points_*.tpl	INIT2PEST	PEST	Template file used to create pilot points data files
real2mod_points_*.in	PEST or PARCALC	REAL2MOD	Input file to shift and un-log-transform the fields in preparation for MODFLOW
real2mod_points_*.tpl	INIT2PEST	PEST	Template files for creating REAL2MOD input files
r???_init.bpa	PEST	PARREP	Contains the final, best parameter values from the SVD-assisted run
r???_init.par	PEST	N/A	Initial parameter values after single Jacobian run
r???_init.pst	INIT2PEST	PEST, SVDAPREP	PEST run control file
r???_init.{rec,res,sen,seo}	PEST	SVDAPREP	PEST run record files
r???_once.pst	INIT2PEST	PARREP	Initial run control file for PEST
r???_svda.pst	SVDAPREP	PEST	PEST run control file for middle SVD iterations
r???_svda.{rec,res,sen,seo}	PEST	N/A	PEST run record files

File saved	Created by	Used by	Brief description
parcalc.tpl	*PEST/SVD*	*PARCALC*	*Intermediate file, not saved, but of importance during PEST's SVD assisted operation. PARCALC converts super parameters to real parameters*
r???_last.pst	PARREP	PEST	Final, best parameter PEST run to get fields and outputs
r???_last.{rec,res,sen,seo}	PEST	N/A	PEST run record files
r???_last.par	PEST	N/A	Final, best parameters file

Additional PEST output files are also saved, but are unused in the analysis at this time.

Table F-4. List of data sources for graphics and tables in this document.

Figure/Table	CVS Data Source under CVS://Tfields *Creation Process*
Figure 3-1	Inputs/scripts/init2pest.py
Figure 3-2	Inputs/data/zone_*.geo, Inputs/config/spec_wells.crd *Basic platting package*
Figure 3-3	Outputs/r???/modeled_zones_points_T.inf *Basic plotting package*
Figure 3-4	Outputs/r???/modeled_zones_points_S.inf *Basic plotting package*
Figure 3-5	Outputs/r???/modeled_zones_points_R.inf *Basic plotting package*
Figure 3-6	Inputs/data/init_bnds.inf *Basic plotting package*
Figure 3-7	Outputs/r???/modeled_points_T.dat *Basic plotting package*
Figure 3-8	Outputs/r???/modeled_points_A.dat *Basic plotting package*
Figure 3-9	Outputs/r???/modeled_points_S.dat *Basic plotting package*
Figure 3-10	Outputs/r???/modeled_points_R.dat *Basic plotting package*
Table 3-1	Inputs/config/spec_points_{T,A,S,R}.in
Figure 3-11	Inputs/data/init_head.mod *Basic plotting package*
Table 3-2	Inputs/config/spec_variogram.str
Figure 3-12	Inputs/scripts/model.sh
Figure 4-1	*N/A*
Figure 4-2	Inputs/scripts/template_pmaster.sh
Figure 4-3	Inputs/scripts/setup10.sh
Table 4-1	Analysis/DataFiles/Tfields-7_14_09.xlsx *Basic Excel plot*
Table 4-2	Analysis/DataFiles/Tfields-7_14_09.xlsx *Basic Excel plot*
Figure 4-4	Analysis/DataFiles/Tfields-7_14_09.xlsx *Basic Excel plot*
Figure 5-1	Analysis/DataFiles/Tfields-7_14_09.xlsx *Data taken from last line of* Results/tracks/r???_travel_LWB.dat, *basic Excel plot*
Figure 5-2	Results/tracks/r???_travel_LWB.dat *Basic plotting package*
Figure 5-3	Results/tracks/r???_travel_domain.dat *Basic platting package*

Figure/Table	CVS Data Source under CVS://Tfields *Creation Process*
Table 5-1	Results/K_fields/r???_K_field.mod *Multiplied by thickness (7.75m), log transformed, then averaged by cell across the 100 final fields.*
Figure 5-4	Results/K_fields/r???_K_field.mod, Results/A_fields/r???_A_field.mod *Multiplied by thickness (7.75m), log transformed, combined resulting T & A (T + ½ A), mean value by cell*
Figure 5-5	Results/K_fields/r???_K_field.mod, Results/A_fields/r???_A_field.mod *Multiplied by thickness (7.75m), log transformed, combined resulting T & A (T + ½ A), standard deviation by cell*
Figure 5-6	Outputs/r???/modeled_init_T.mod *Values compared against **log10T >= -5.41** and summed by cell*
Figure 5-7	Results/K_fields/r???_K_field.mod, Results/A_fields/r???_A_field.mod *Multiplied by thickness (7.75m), log transformed, combined resulting T & A (T + ½ A), values compared against **log10T >= -5.41** and summed by cell*
Figure 5-8	*Difference between results in Figure 5-6 ond Figure 5-7, positive changes only*
Figure 5-9	*Difference between results in Figure 5-6 and Figure 5-7, negative changes only*
Figure 5-10	Results/A_fields/r???_A_field.mod *Average log values by cell*
Figure 5-11	Results/A_fields/r???_A_field.mod *Standard deviation of log values by cell.*
Figure 5-12	Results/S_fields/r???_S_field.mod *Multiplied by thickness, mean of log values by cell*
Figure 5-13	Results/S_fields/r???_S_field.mod *Multiplied by thickness, standard deviation of log values by cell*
Figure 5-14	Results/R_fields/r???_R_field.mod *Log transformed, converted to meters per year*
Figure 5-15	Results/R_fields/r???_R_field.mod *Log transformed*
Figure 5-16	Outputs/{r???,Update/r???,Update2/r???}/modeled_head.smp *Depending on the field selected, the results were used from the correct directory*
Figure 5-17	Outputs/{r???,Update/r???,Update2/r???}/modeled_head.smp *Depending on the field selected, the results were used from the correct directory*
Figure A-1 thru 6	*Outputs from Listing A-1. Resulting data file:* Inputs/data/init_head.mod
Figure B-1	Analysis/DataFiles/T_Wells_UTMNAD27.xls
Figure B-2	Analysis/DataFiles/T_Wells_UTMNAD27.xls *VarioWin was used to create the image*
Table B-3	Inputs/config/spec_variogram.str
Figure B-3	Analysis/DataFiles/T_Wells_UTMNAD27.xls *VarioWin was used to create the image*
Table C-1	Inputs/data/meas_head.smp (revision 1.2)
Figure C-1 through Figure C-10	Outputs/{r???,Update/r???,Update2/r???}/modeled_head.smp *Depending on the field selected, the results were used from the correct directory, graphics use standard box-and-whisker plotting functions*

Figure/Table	CVS Data Source under CVS://Tfields *Creation Process*
Figures in D.1	Outputs/{r???,Update/r???,Update2/r???}/modeled_h3.smp, Inputs/data/meas_h3.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.2	Outputs/{r???,Update/r???,Update2/r???}/modeled_wipp13.smp, Inputs/data/meas_wipp13.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.3	Outputs/{r???,Update/r???,Update2/r???}/modeled_h11.smp, Inputs/data/meas_h11.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.4	Outputs/{r???,Update/r???,Update2/r???}/modeled_p14.smp, Inputs/data/meas_p14.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.5	Outputs/{r???,Update/r???,Update2/r???}/modeled_h19.smp, Inputs/data/meas_h19.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.6	Outputs/{r???,Update/r???,Update2/r???}/modeled_wqsp1.smp, Inputs/data/meas_wqsp1.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.7	Outputs/{r???,Update/r???,Update2/r???}/modeled_wqsp2.smp, Inputs/data/meas_wqsp2.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.8	Outputs/{r???,Update/r???,Update2/r???}/modeled_wipp11.smp, Inputs/data/meas_wipp11.smp *Plotted drawdown vs. date for each observation well separately*
Figures in D.9	Outputs/{r???,Update/r???,Update2/r???}/modeled_sn114.smp, Inputs/data/meas_sn114.smp *Plotted drawdown vs. dote for each observation well separately*
Appendix E	Analysis/DataFiles/Tfields-7_14_09.xlsx

Appendix G: Files Used – Listings, Source Code, and Validation Processes

This is not a complete listing of all configuration files used during the calibration – please see the CVS repository to obtain all the files. However, the scripts that were written by the analysts for this task, and the configuration files that controlled those scripts are presented here to keep them with the analysis report.

Report comments may be inserted into the code for clarity, and will be in *red-italic-serif* font. *These comments do not appear in the files in CVS.* A narrative of how the code works, and a description of how the codes were/can be verified will be provided before each listing.

G.1 Configuration Codes

The codes in this section represent the scripts run during the configuration phase (pre-calibration) by the Run Control system.

G.1.1 Listing – CVS://Tfields::Inputs/scripts/setup.sh

This script creates the inputs and runs the INIT2PEST script (described in G.1.2). Verification is provided in the log files, and mainly in whether or not a calibration actually launches. If it launches, then this script worked; if it does not, then this script failed.

```
#!/bin/bash
# $Id: setup.sh,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
# $Source: /nfs/data/CVSLIB/Tfields/Inputs/scripts/setup.sh,v $
#
# This script creates the necessary files to run a Culebra T Field
# inverse calibration process

# Get command line input parameters
FIELD=$1

# Require that a base field be supplied on the command line
[ -z "$FIELD" ] && {
    echo "You MUST specify a base field ID"
    exit
}

FIELD=${FIELD//r/}
FIELD=${FIELD//cood/}
PESTFILE=r${FIELD}
FIELD=r${FIELD}coord
cp -f ${FIELD}.map modeled_basefield.map
echo ${FIELD//coord} > modeled_fieldID.tex

rm -f modeled_*factors*.bin

The next line launches the INIT2PEST script, which performs the majority of the setup process
./init2pest.py --pest-case=${FIELD//coord} spec_setup.in

PESTINIT="${PESTFILE}_init"
PESTSVDA="${PESTFILE}_svda"
PESTLAST="${PESTFILE}_last"
PESTONCE="${PESTFILE}_once"

NSLAVES=6
MODELCMD=model
MODELINIT=./${MODELCMD}.sh
N=1
```

```
The next two sections set the current directory first in precedence, then find the appropriate executables or scripts and place them in variables
export PATH=.:$PATH

PPEST=`which ppest.exe`
PSLAVE=`which pslave.exe`
PEST=`which pest.exe`
SVDAPREP=`which svdaprep.exe`
PARREP=`which parrep.exe`
PPK2FAC=`which ppk2fac.exe`
FAC2REAL=`which fac2real.exe`
REAL2MOD=./real2mod.py
MODFLOW=`which mf2k.exe`
MOD2OBS=`which mod2obs.exe`
OBS2REAL=./obs2real.py
PICALC=`which picalc.exe`
PARCALC=`which parcalc.exe`
DTRKMF=`which dtrkmf.exe`

The following lines use "sed" regular expressions to replace variables with full path names for individual executables to assure we are using the
correct, QA'd versions of the files
while [[ $N -le $NSLAVES ]]
  do
  # Create the regular expression file to make a script from template
  echo "s/slave1/slave${N}/" > modeled_pslave.sed
  echo "s/FIELDID/${FIELD}S/" >> modeled_pslave.sed
  echo "s@PSLAVE@${PSLAVE}@g" >> modeled_pslave.sed
  echo "s@MODELINIT@${MODELINIT}@g" >> modeled_pslave.sed
  echo "s@PPK2FAC@${PPK2FAC}@g" >> modeled_pslave.sed
  echo "s@FAC2REAL@${FAC2REAL}@g" >> modeled_pslave.sed
  echo "s@REAL2MOD@${REAL2MOD}@g" >> modeled_pslave.sed
  echo "s@MODFLOW@${MODFLOW}@g" >> modeled_pslave.sed
  echo "s@MOD2OBS@${MOD2OBS}@g" >> modeled_pslave.sed
  echo "s@OBS2REAL@${OBS2REAL}@g" >> modeled_pslave.sed
  echo "s@PICALC@${PICALC}@g" >> modeled_pslave.sed
  echo "s@PARCALC@${PARCALC}@g" >> modeled_pslave.sed
  sed -f modeled_pslave.sed template_pslave.sh > pest_slave.sh
  rm -f modeled_pslave.sed
  # Submit the new script file to the queue
  [ -d slave${N} ] || {
      mkdir slave${N}
  }
  FILES=`cat spec_slavefiles.lst`
  cp -f -p -u $FILES slave${N}
  echo "Copied files to slave${N}"
  N=$((N+1))
done

echo "s/Master/${FIELD}M/" >> modeled_pmaster.sed
echo "s@PPEST@${PPEST}@g" >> modeled_pmaster.sed
echo "s@SPEST@${PEST}@g" >> modeled_pmaster.sed
echo "s@SVDAPREP@${SVDAPREP}@g" >> modeled_pmaster.sed
echo "s@PARREP@${PARREP}@g" >> modeled_pmaster.sed
echo "s@PPK2FAC@${PPK2FAC}@g" >> modeled_pmaster.sed
echo "s@PESTINIT@${PESTINIT}@g" >> modeled_pmaster.sed
echo "s@PESTSVDA@${PESTSVDA}@g" >> modeled_pmaster.sed
echo "s@PESTLAST@${PESTLAST}@g" >> modeled_pmaster.sed
echo "s@PESTONCE@${PESTONCE}@g" >> modeled_pmaster.sed
echo "s@PESTHALF@${PESTHALF}@g" >> modeled_pmaster.sed
echo "s@PESTSVD2@${PESTSVD2}@g" >> modeled_pmaster.sed
echo "s@DTRKMF@${DTRKMF}@g" >> modeled_pmaster.sed
echo "s@NPAR@${NPAR}@g" >> modeled_pmaster.sed
sed -f modeled_pmaster.sed template_pmaster.sh > pest_master.sh
```

G.1.2 Listing - CVS://Tfields::Inputs/scripts/init2pest.py

```
#!/usr/bin/python
# $Id: init2pest.py,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
# $Author: dbhart $
#
# Read a MODFLOW array, preform some mathematical transforms, then
# write back to a formatted MODFLOW array file

import sys, pestutil
from sys import stdin, stdout, stderr
from optparse import OptionParser

The main program parses the command line arguments, then launches the subroutines to create observation and parameter data from the
configuration data. It then builds the PEST configuration files. The "pestutil" namespace refers to the pestutil.py model described in G.4.1
def main( parser ):
    (options, args) = parser.parse_args()
    if args is not None:
        for filename in args:
            argfile = open(filename, 'r')
            alist = []
            for arg in argfile:
                aa = arg.split()
                for a in aa: alist.append(a.strip())
            (options, args) = parser.parse_args(alist,options)
    (obsData,obsGrps,obsFiles) = pest_obs_data(options)
    (parData,parGrps,parFiles) = pest_par_data(options)
    pestutil.build_pest_file(options,'init',
                    obsData,obsGrps,obsFiles,
                    parData,parGrps,parFiles )
    pestutil.build_pest_file(options,'once',
                    obsData,obsGrps,obsFiles,
                    parData,parGrps,parFiles, 0)
    pestutil.build_pest_rmf(options,'init')
    pestutil.build_pest_rmf(options,'svda')

The following subroutine creates observation data and INS files for PEST
def pest_obs_data(options):
    peststub = options.peststub
    obsOptfile = options.obsOptfile
    print ' Processing observation data'
    print '  Reading from "'+obsOptfile+'"'
    (obsData,obsGrps,obsFiles) = pestutil.create_observation_data(obsOptfile,options)
    return obsData, obsGrps, obsFiles

The following subroutine creates the parameters, creating zones, placing pilot points, and then finding initial values
def pest_par_data(options):
    peststub = options.peststub
    parOptfiles = options.parOptfiles
    parData = parGrps = parFiles = []
    for (param,optfile) in parOptfiles:
        print ' For the parameter',param+':'
        print '  Reading from "'+optfile+'"'
        Parameters = pestutil.ParameterOptions(param)
        Parameters.option_parser(optfile,options)
        Parameters.create_zones(options)
        Parameters.create_initial_field(options)
        Parameters.create_pilot_points(options)
        Parameters.create_utility_inputs(options)
        Parameters.print_latex_table()
        (pData,pGrps,pFiles) = Parameters.get_pest_parameters(options)
        parData = parData + pData
        parGrps = parGrps + pGrps
        parFiles = parFiles + pFiles
    return parData, parGrps, parFiles


The command line parser is defined here. The "help" metavariable describes the options, and the "dest" metavariable is the field in the
"options" structure that is returned and used in the above subroutines
def init2pest_parser( ):
    parser = OptionParser()
    parser.add_option("--pest-case",dest="peststub",metavar="FILE",default='culebra',
                      help="The prefix for the PEST control files to create (will create
FILE_init.pst and FILE_once.pst)")
```

```
    parser.add_option("--observations-options-file",dest="obsOptfile",metavar="FILE",
                      help="The file containing the options for the observation data creation")
    parser.add_option("--parameter-options-file",dest="parOptfiles",action="append",metavar="PAR
FILE",nargs=2,
                      help="The PAR parameter and the FILE where the parameter options are
listed")
    parser.add_option('--zones-filename-format',dest='zonefmt')
    parser.add_option('--pilotpoints-filename-format',dest='ppffmt')
    parser.add_option('--factors-filename-format',dest='facfmt')
    parser.add_option('--initial-values-field-filename-format',dest='initfmt')
    parser.add_option('--kriged-parameter-field-filename-format',dest='krigfmt')
    parser.add_option('--final-parameter-field-filename-format',dest='modfmt')
    parser.add_option('--flow-field-filename-format',dest='flowfmt')
    parser.add_option('--modflow-output-filename-format',dest='mf2kfmt')
    parser.add_option('--modeled-samples-filename-format',dest='modlfmt')
    parser.add_option('--measured-samples-filename-format',dest='measfmt')
    parser.add_option('--command-redirect-filename-format',dest='inpredfmt')
    parser.add_option('--num-slaves',dest='nslaves',default=6,type='int')
    parser.add_option('--model-command',dest='model',default='./model.sh')
    return parser


if __name__ == "__main__":
    parser = init2pest_parser()
    main(parser)
```

G.2 Execution Codes

G.2.1 Listing – CVS://Tfields::Inputs/scripts/runpest.sh

This code runs PEST, and is called by the Run Control script after setup is complete. The first parameter is the field ID, the second is the number of slaves. This script either works or immediately fails (and no results are created).

```
#!/bin/bash

NSLAVES=$2
FIELD=$1

[ -z "$FIELD" ] && {
    FIELD=`ls -t *_once.pst | head -n 1`
    FIELD=${FIELD//_once.pst}
}

[ -z "$NSLAVES" ] && {
    NSLAVES=6
}

The following ensures that the field ID is in the correct format (just a number) before changing it back to full ID
FID=${FIELD//r/}
FID=${FID//cood/}
FIELD=r${FID}coord

echo -n "Starting in 3 . "
sleep 1
echo -n "2 . "
sleep 1
echo -n "1 . "
sleep 1
echo "done!"

The "pest_slave.sh" file is created by setup.sh from template_pslave.sh – see G.2.2
qsub -t 2-$NSLAVES pest_slave.sh 2> ${FIELD}.err | tee -a ${FIELD}.out
The "pest_master.sh" file is created by setup.sh from template_pmaster.sh – see G.2.3
qsub -sync yes pest_master.sh 2> ${FIELD}.err | tee -a ${FIELD}.out

qdel -f ${FIELD}S
rm -rf slave?
```

G.2.2 Listing - CVS://Tfields::Inputs/scripts/template_pslave.sh

This script launches PEST's PSLAVE executable on the client machines on the cluster. If this script runs correctly, PEST works. Otherwise, PEST fails; this information is captured in the PEST run record (*.rec).

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N FIELDID
#$ -j y
##$Id: template_pslave.sh,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
##$Source: /nfs/data/CVSLIB/Tfields/Inputs/scripts/template_pslave.sh,v $

export MYSLAVE=slave$SGE_TASK_ID
FILES=`cat spec_slavefiles.lst`

# Setup slave directory
mkdir $MYSLAVE
cp -f $FILES $MYSLAVE
cd $MYSLAVE

# First PEST run
rm -f model.svda

PSLAVE <<EOF
MODELINIT
EOF

# First SVDAPEST run
echo "true" > model.svda
sleep 10

PSLAVE <<EOF
MODELINIT
EOF

# Clean up slave directory
cd ..
rm -rf $MYSLAVE
```

G.2.3 Listing – CVS://Tfields::Inputs/scripts/template_pmaster.sh

This script launches PEST's main program, PPEST, on the cluster. It also runs SVDAPREP and PARREP to create the secondary PEST runs. This script is captured in Figure 4-2. Failure of this script is clear in any log file, since PEST will produce no results.

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N Master
#$ -j y
# $Id: template_pmaster.sh,v 1.4 2009/06/04 15:45:53 dbhart Exp $
# $Source: /nfs/data/CVSLIB/Tfields/Inputs/scripts/template_pmaster.sh,v $

export MYSLAVE=slave1
FILES=`cat spec_slavefiles.lst`
# Setup slave directory
mkdir $MYSLAVE
cp -f $FILES $MYSLAVE
cd $MYSLAVE

The following lines run the first Jacobian calculation PEST run
# First PEST run
rm -f model.svda

PSLAVE <<EOF &
MODELINIT
EOF

cd ..

PPEST PESTINIT

The following lines of code set up the SVD assisted PEST run
NUMPAR=`tail -n 1 PESTINIT.svd | awk '{print $9}'`

SVDAPREP <<EOF
PESTINIT.pst
$NUMPAR
PESTSVDA.pst
10
0.1
s
EOF

sleep 10

SLAVES=`ls -d slave*`
for Slave in $SLAVES
  do
  echo "true" > $Slave/model.svda
done

cd $MYSLAVE

# First SVDAPEST run
echo "true" > model.svda
sleep 10

PSLAVE <<EOF &
MODELINIT
EOF

# Clean up slave directory
cd ..

PPEST PESTSVDA

The followig lines set up the final PEST run with the best parameters
PARREP PESTINIT.bpa PESTONCE.pst PESTLAST.pst

sleep 10
```

```
SPEST PESTLAST
kill -TERM `jobs -p`
rm -rf $MYSLAVE
```

G.3 Forward Model Codes

The following scripts are used in the forward model, and are for each parameter (or super-parameter) during each iteration of PEST. Failure of these codes is detected automatically by PEST, resulting in an early termination to PEST, obvious in the log files.

G.3.1 Listing – CVS://Tfields::Inputs/scripts/model.sh

This script represents Figure 4-1, and runs all the necessary pre- and post-MODFLOW codes.

PARCALC (if an SVD run) → PPK2FAC (if files do not exist) → FAC2REAL → REAL2MOD → MODFLOW → MOD2OBS → OBS2REAL → DTRKMF (if running a final transport run)

```
#!/bin/bash
#$Id: model.sh,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
echo Created on Thu Aug  7 16:11:18 MDT 2008 by create-model.sh
PARAMETERS=`cat spec_parameters.lst`
TESTS=`cat spec_steady.lst spec_transient.lst`
STDERR=model.err
STDOUT=model.log
#STDERR=/dev/stdout
#STDOUT=/dev/stdout

run_model() {
# Clean the output files and start model
  echo > $STDOUT
  echo > $STDERR
  echo -n "MODEL::"
  ulimit -S -t 120

  # IF this is an SVD assisted run:
  [ -s model.svda ] && {
      # Remove previous parameter estimates
      for P in $PARAMETERS
        do
        /bin/rm modeled_points_${P}.out 1>> $STDOUT 2>> $STDERR
      done

      # Create parameters from super parameters
      ./parcalc.exe 1>> $STDOUT 2>> $STDERR
      echo -n PARCALC:

      # Creating prior information calculation (if applicable)
      [ -s picalc.tpl ] && {
          ./picalc.exe 1>> $STDOUT 2>> $STDERR
          echo -n PICALC:
      }
  }

  # Now creating fields
  for P in $PARAMETERS
    do
    # Create kriging factors if they are missing
    [ -s modeled_factors_points_${P}.bin ] || {
        echo -n PPK2FAC[points_${P}].
        ./ppk2fac.exe < ppk2fac_points_${P}.in 1>> $STDOUT 2>> $STDERR
    }

    # Use pilot points and kriging factors to get $\log_{10}$ field
    echo -n FAC2REAL[points_${P}].
    ./fac2real.exe < fac2real_points_${P}.in 1>> $STDOUT 2>> $STDERR

    # Convert $\log_{10}$ field into real space
    echo -n REAL2MOD[points_${P}].
    ./real2mod.py real2mod_points_${P}.in 1>> $STDOUT 2>> $STDERR
  done

  for T in $TESTS
```

```
    do
    # Now run MODFLOW for each test
    echo -n MODFLOW[${T}].
    ./mf2k.exe mf2k_${T}.nam 1>>$STDOUT 2>MODFLOW.ERR
    [ -s MODFLOW.ERR ] && {
        echo -n MODFLOW[${T}.pcg].
        ulimit -S -t unlimited
        ./mf2k.exe mf2k_${T}.pcg.nam 2>>$STDOUT 2>MODFLOW.ERR
        ulimit -S -t 120
    }
    rm -f MODFLOW.ERR

    # Get the observations from the test results
    echo -n MOD2OBS[${T}].
    ./mod2obs.exe < mod2obs_${T}.in 1>>$STDOUT 2>>$STDERR

    # Putting results from tests into single column format
    echo -n OBS2REAL[${T}].
    ./obs2real.py obs2real_${T}.in #1>>$STDOUT 2>>$STDERR
  done

  [ -s model.travel ] && {
  # Runing DTRTMF to LWB
      echo -n DTRKMF[WIPP].
      [ -s fort.33 ] || {
          ln -f elev_top.mod fort.33
      }
      [ -s fort.34 ] || {
          ln -f elev_bot.mod fort.34
      }
      [ -s modeled_flow.bud ] && {
          ./dtrkmf.exe dtrkmf_wipplwb.inp modeled_flow.bud modeled_travel_wipplwb.dat
modeled_dtrkmf.dbg 1>>$STDOUT 2>>$STDERR ;
          ./dtrkmf.exe dtrkmf_domain.inp modeled_flow.bud modeled_travel_domain.dat
modeled_dtrkmf.dbg 1>>$STDOUT 2>>$STDERR ;
      }
      tail -n 1 modeled_travel_wipplwb.dat | awk '{print $1}' > modeled_travel.out
      [ -s modeled_travel_wipplwb.dat ] || {
          echo '9999999999' > modeled_travel.out ;
      }

  # Getting Recharge Stats
      echo -n MOD2STAT[recharge].
      ./mod2stat.pl modeled_R_field.mod modeled_recharge.out 1>>$STDOUT 2>>$STDERR
  }
}


ResetTol() {
This function resets tolerances in the MODFLOW solvers
# Reset the solver tolerances
    echo -e '3.0 2.2 5.4 0\n2 50 1.0E-08 1.0 0 ' > mf2k_steady.lmg
    echo -e '3.0 2.2 5.4 0\n2 50 1.0E-08 1.0 1 ' > mf2k_transient.lmg
    echo -e '100 50 2\n1.0e-002 1.0e-08 1 2 0 3 1' > mf2k_transient.pcg
}

RaiseTol() {
Occasionally, MODFLOW cannot converge. In this case, tolerances are lowered slowly until MODFLOW is able to converge
# Rase solver tolerances by one order of magnitude
    NewTol=$(awk < mf2k_steady.lmg 'NR==2 {NT = $3*10;if (NT <= MaxTol) printf("%6.1E", NT)}'
MaxTol=0.01)
    [ "$NewTol" ] || { echo mf2k could not converge ; exit ; }
    echo -e '3.0 2.2 5.4 0\n2 50 '$NewTol' 1.0 1 ' > mf2k_steady.lmg

    NewTol=$(awk < mf2k_transient.lmg 'NR==2 {NT = $3*10;if (NT <= MaxTol) printf("%6.1E", NT)}'
MaxTol=0.01)
    [ "$NewTol" ] || { echo mf2k could not converge ; exit ; }
    echo -e '3.0 2.2 5.4 0\n2 50 '$NewTol' 1.0 1 ' > mf2k_transient.lmg

    NewTol=$(awk < mf2k_transient.pcg 'NR==2 {NT = $2*10;if (NT <= MaxTol) printf("%6.1E", NT)}'
MaxTol=0.01)
    [ "$NewTol" ] || { echo mf2k could not converge ; exit ; }
    echo -e '100 50 2\n1.0e-002 '$NewTol' 1 2 0 3 1 ' > mf2k_transient.pcg
}

# Start of main script

ResetTol
```

```
# Save previous iteration values if necessary
[ -s model.svda ] && {
    ITERNUM=`grep ITERATION ../*_svda.rec | tail -n 1 | awk '{printf("%s",$5)}'`
    PREVNUM=`cat temp_iternum`
    [ "$ITERNUM" == "$PREVNUM" ] || {
        PARAMS=`ls ../*_init.bpa`
        cp $PARAMS ${PARAMS//.bpa}_ITER_${ITERNUM}.bpa
    }
    echo $ITERNUM > temp_iternum
}

run_model
while [ 1 ]
  do
  #  Runtime error handling for convergence failure
  CONVGFAIL=`grep -i "FAILED TO CONVERGE" *.lst`
  if [ -n "$CONVGFAIL" ]
      then
      echo -n -e "Conv_failed_restart"
      # Get the more flexible solver rules to finish run
      RaiseTol
      # Re-run the model
      run_model
  else
      # Put back the good rules when done
      ResetTol
      break
  fi
done
echo
```

G.3.2 Listing - CVS://Tfields::Inputs/scripts/real2mod.py

This code converts the log-space field *modeled_points_?.mod* to the real-space *modeled_?_field.mod*. This code was verified by examination of input and outputs for r109. This code also shifts the fields by a certain amount, if so indicated in the input file. The input options are provided in the parser, and the "help" metavariable describes what those options are. This script makes use of the PESTUTIL module presented in G.4.1.

```
#!/usr/bin/python
# $Id: real2mod.py,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
# $Author: dbhart $
#
# Read a MODFLOW array, preform some mathematical transforms, then
# write back to a formatted MODFLOW array file

import pestutil, sys
from sys import stdin, stdout, stderr
from optparse import OptionParser

def main( parser=None ):
    "parse the command line options and/or options file"
    infile = None
    outfile = None
    shift = 0.0
    scale = 1.0
    maskShft = 0.0
    xform = ""
    if parser:
        (options, args) = parser.parse_args()
        alist = []
        if args:
            argfile = open(args[0], 'r')
            for arg in argfile:
                alist.append(arg.rstrip())
            (options, args) = parser.parse_args(args=alist, values=options)
        (options, args) = parser.parse_args(values=options)
        infile = options.infile
        infileformat = options.infileformat
        outfile = options.outfile
        outfileformat = options.outfileformat
        maskfile = options.maskFile
        maskfileformat = options.maskfileformat
        maskSymb = options.maskValue
        if options.maskShift: maskShft = options.maskShift
        if options.myShift: shift = options.myShift
        if options.myScaleUp: scale = options.myScaleUp
        if options.myScaleDown: scale = 1.0 / options.myScaleDown
        if options.transform: xform = options.transform
    if infile is None:
        print >> stderr, 'Which file would you like to transform? ',
        infile = str.strip(stdin.readline())
    if outfile is None:
        print >> stderr, 'What file would you like to produce? ',
        outfile = str.strip(stdin.readline())
    transforms = ( shift, scale , maskShft, xform.upper() )
    data = pestutil.FieldData()
    data.load_field(infile,infileformat)
    if maskSymb:
        data.load_mask(maskfile,maskfileformat)
        data.doMask = maskSymb
    data.run_transforms(transforms)
    data.save_field(outfile,outfileformat)

def mf_parser( ver="N/A" ):
    ver = "%prog " + ver
    parser = OptionParser(version=ver)
    parser.add_option("-i", "--input", dest="infile",
                      help="MODFLOW-2000 formatted FILE to transform", metavar="FILE")
    parser.add_option("-o", "--output", dest="outfile",
                      help="MODFLOW-2000 formatted FILE to write", metavar="FILE" )
```

```
    parser.add_option("-g", "--gridspec", dest="specfile",
                      help="domain grid specification file", metavar="FILE")
    parser.add_option("--mask-value", dest="maskValue", type="float",
                      help="set MASK in the input file", metavar="MASK")
    parser.add_option("--mask-file", dest="maskFile",
                      help="replace MASK with values from FILE", metavar="FILE")
    parser.add_option("--mask-shift", dest="maskShift", type="float",
                      help="shift MASK replacement values by MS", metavar="MS")
    parser.add_option("--shift", dest="myShift", type="float",
                      help="shift all values (including masked values) by SS [done in transformed
space]", metavar="SS" )
    parser.add_option("--scale-up", dest="myScaleUp", type="float",
                      help="scale all values up by SU [done in real-space]", metavar="SU" )
    parser.add_option("--scale-down", dest="myScaleDown", type="float",
                      help="scale all values down by SD [done in real-space]", metavar="SD")
    parser.add_option("--transform", dest="transform",
                      type="choice", choices=("log10","pow10"),
                      help="transform values" )
    parser.add_option("--input-format", dest="infileformat",
                      type="choice", choices=("MODFLOW","GEOEAS") )
    parser.add_option("--output-format", dest="outfileformat",
                      type="choice", choices=("MODFLOW","GEOEAS","GNUPLOT") )
    parser.add_option("--maskfile-format", dest="maskfileformat",
                      type="choice", choices=("MODFLOW","GEOEAS") )
    parser.set_defaults(myShift=0.0, myScaleUp=1.0, myScaleDown=1.0,
                        maskShift=0.0,
                        infileformat="MODFLOW",
                        outfileformat="MODFLOW",
                        maskfileformat="MODFLOW")
    return parser

if __name__ == "__main__":
    parser = mf_parser('1.0')
    main(parser)
```

G.3.3 Listing – CVS://Tfields::Inputs/scripts/obs2real.py

This code converts the outputs from the PEST program MOD2OBS to a more easy to work with format. This code was verified by examining the *modeled_?.smp* and *modeled_?.out* files to make sure they were the same. The input options are provided in the parser, and the "help" metavariable describes what those options are. This script makes use of the PESTUTIL module presented in G.4.1.

```
#!/usr/bin/python
# $Id: obs2real.py,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
# $Author: dbhart $
#
# Read a MODFLOW array, preform some mathematical transforms, then
# write back to a formatted MODFLOW array file

import pestutil, sys
from sys import stdin, stdout, stderr
from optparse import OptionParser

def main( parser=None ):
    "parse the command line options and/or options file"
    measfile = None
    modlfile = None
    outstub = None
    pumpID = None
    crdfile = None
    startDate = None
    if parser:
        (options, args) = parser.parse_args()
        alist = []
        if args:
            argfile = open(args[0], 'r')
            for arg in argfile:
                alist.append(arg.rstrip())
            (options, args) = parser.parse_args(args=alist, values=options)
        (options, args) = parser.parse_args(values=options)
        measfile = options.measfile
        modlfile = options.modlfile
        pumpID = options.pumpID
        outstub = options.output
        startDate = options.startdate
        b_singleGroup = options.b_singleGroup
    if outstub is None:
        mdf = modlfile.split('.')
        del mdf[-1]
        outstub = '.'.join(mdf)
    The following creates a SampleData object for the give pumping test ID and start-date, then loads measured and modeled data
    data = pestutil.SampleData(pumpID,startDate)
    data.load_measured(measfile)
    data.load_modeled(modlfile)
    data.compare_samples()
    data.print_outfile(outstub,b_singleGroup)
    if options.ssefile is not None:
        data.print_summary_data(options.ssefile,options.sseappend)
    if options.fulldata is not None:
        data.print_full_data(options.fulldata)


def obs2real_parser( ver="N/A" ):
    ver = "%prog " + ver
    parser = OptionParser(version=ver)
    parser.add_option("--measured", dest="measfile", metavar="FILE",
                      help="Bore sample file of true field measurements")
    parser.add_option("--modeled", dest="modlfile", metavar="FILE",
                      help="Bore sample file of modeled results")
    parser.add_option("--coords", dest="crdfile", metavar="FILE",
                      help="Well coordinates file")
    parser.add_option("-d","--start-date", dest="startdate", metavar="DATETIME",
                      help="Date and time of the start of pumping")
    parser.add_option("-o","--output-stub", dest="output", metavar="FILESTUB",
                      help="[directory] and filename stub (w/o extension) for output")
    parser.add_option("--full-listing-file", dest="fulldata", metavar="FILE",
```

```
                        help="Use to output full data as well as truncated residuals")
    parser.add_option("--summary-file", dest="ssefile", metavar="FILE",
                        help="File to write summary information into")
    parser.add_option("-a","--summary-append", dest="sseappend", action="store_true",
                        help="Append to, rather than create, summary information file")
    parser.add_option("-p","--pump-id", dest="pumpID", metavar="WELL",
                        help="Pumping well identifier")
    parser.add_option("-s","--steady-state", dest="b_singleGroup", action="store_true",
                        help="If all wells are in a single group, use this flag")
    parser.set_defaults(sseappend=False, b_singleGroup=False)
    return parser


if __name__ == "__main__":
    parser = obs2real_parser('1.0')
    main(parser)
```

G.4 Main PEST utility module

This utility module contains the meat of the INIT2PEST, REAL2MOD, and OBS2REAL codes. Most of the code in this module regards file creation, and incorrect syntax results in a complete failure of PEST, so if PEST runs, the code works. Other verification methods will be called out in the code comments – most are simply the visual inspection of the resulting fields (such as examining Figures 3-4 and 3-7) or through the output to the console that is captured by the log files (see Section H.4, for example of INIT2PEST output). The particular program that uses a function is indicated at the beginning of the comment

G.4.1 Module – CVS://Tfields::Inputs/scripts/pestutil.py

```
# $Id: pestutil.py,v 1.1.1.1 2008/11/05 23:57:43 dbhart Exp $
# $Module: pestutil $
# $Author: dbhart $
#
# This module supports MODFLOW-2000 python utility functions, simplified Geo-EAS
# file functions, and PEST Groundwater Utilitiy file functions
#
# In this module, the DATA object is always 0-indexed, but otherwise the same as
# MODFLOW-2000 format -- i.e., Node 0 is the upper left cell on the topmost layer
# moving row-major across the layer, then to the next lower layer

import sys, datetime, time, math
from optparse import OptionParser

########################################
# PEST Configuration Routines

INIT2PEST: This routine takes observation data read in from the files specified in "spec_observations.in" and creates the observation groups
and observation values used in the PEST configuration file. Manual examination of one complete PEST file is sufficient to verify proper function.
and PEST would fail to launch if this data was incorrectly specified.
def create_observation_data( filename , baseOpts=None ):
    parser = observation_data_parser()
    AllObsGroups = []
    AllObsData = []
    AllInsPairs = []
    argfile = open(filename, 'r')
    alist = []
    for arg in argfile:
        alist.append(arg.strip())
    (options, args) = parser.parse_args(args=alist)
    ststate = options.ststate
    SSDict = {}
    StartDates = {}
    for test in options.ststate:
        SSDict[test.upper()] = True
    try:
        (StartDates, PumpList) = read_pumping(options.pumpfile)
    except:
        print 'the pumping file is invalid'
        raise
    if options.trtests == []: trtests = PumpList
    else: trtests = options.trtests
    alltests = ststate + trtests
    measFiles = {}
    modlFiles = {}
    if baseOpts is not None:
        for test in alltests:
            if baseOpts.modlfmt is not None:
                modlFile = baseOpts.modlfmt%{'test': test.lower() }
                modlFiles[test.upper()] = modlFile
            if baseOpts.measfmt is not None:
                measFile = baseOpts.measfmt%{'test': test.lower() }
                measFiles[test.upper()] = measFile
    for (test,filename) in options.measFile:
        measFiles[test.upper()] = filename
    for (test,filename) in options.modflowFile:
        modlFiles[test.upper()] = filename
```

```
    for test in alltests:
        if test.upper() in SSDict: b_steadyState = True
        else: b_steadyState = False
        if test.upper() in measFiles: msFile = measFiles[test.upper()]
        else: msFile = 'meas_'+test.lower()+'.smp'
        if test.upper() in modlFiles: mfFile = modlFiles[test.upper()]
        else: mfFile = 'modeled_'+test.lower()+'.bin'
        if not b_steadyState: startdate = StartDates[test.upper()]
        else: startdate = None
        data = SampleData(test.upper(),startdate)
        data.load_measured(msFile)
        obsGroups = data.assign_obs_groups(b_steadyState)
        obsData = data.assign_obs_data(b_steadyState)
        print '   For test id',test.upper()
        print '    Added',len(obsData),'observations in',len(obsGroups),'groups'
        AllObsGroups = AllObsGroups + obsGroups
        AllObsData = AllObsData + obsData
        outFile = mfFile.split('.')
        insFile = mfFile.split('.')
        outFile[-1] = 'out'
        insFile[-1] = 'ins'
        outFile = '.'.join(outFile)
        insFile = '.'.join(insFile)
        write_ins_file(insFile,obsData)
        print '    saved observation instruction file to:',insFile
        AllInsPairs.append( (insFile,outFile,) )
    return AllObsData, AllObsGroups, AllInsPairs

INIT2PEST:This procedure parses the "spec_observations.in" file for options.
def observation_data_parser( ):
    parser = OptionParser()
    parser.add_option("-s","--steady-state-test",dest="ststate",action="append",
                      metavar="TestID",help="Steady state test ID. May be used more than once")
    parser.add_option("--pumping-test-file",dest="pumpfile",metavar="FILE",
                      help="PEST GW Utility Pumping File. All pumping test start dates should be
in this file, and bore names must match those listed as pumping tests")
    parser.add_option("-p","--pumping-test",dest="trtests",action="append",
                      metavar="TestID",help="Pumping tests to use. If none are specified, all
tests listed in the pumping-test-file will be used.")
    parser.add_option("--use-test-measured-
file",dest="measFile",metavar="TEST,FILE",action="append",nargs=2,
                      help="If unspecified for a given test, this utility assumes data is in a
file of format 'meas_TESTID.smp'")
    parser.add_option("--use-test-modflow-output-
file",dest="modflowFile",metavar="TEST,FILE",action="append",nargs=2,
                      help="If unspecified for a given test, this utility assumes data is output
to a file of format 'modeled_TESTID.bin'")
    parser.set_defaults(ststate=[],pumpfile="",trtests=[],measFile=[],modflowFile=[])
    return parser

COMMENT: This parse reads command values, however, it is not currently used
def commands_parser( ):
    parser = OptionParser()
    parser.add_option('--pest',dest='pest',default='./pest.exe')
    parser.add_option('--ppest',dest='ppest',default='./ppest.exe')
    parser.add_option('--pslave',dest='pslave',default='./pslave.exe')
    parser.add_option('--parcalc',dest='parcalc',default='./parcalc.exe')
    parser.add_option('--picalc',dest='picalc',default='./picalc.exe')
    parser.add_option('--parrep',dest='parrep',default='./parrep.exe')
    parser.add_option('--ppk2fac',dest='ppk2fac',default='./ppk2fac.exe')
    parser.add_option('--fac2real',dest='fac2real',default='./fac2real.exe')
    parser.add_option('--real2mod',dest='real2mod',default='./real2mod.py')
    parser.add_option('--mf2k',dest='mf2k',default='./mf2k.exe')
    parser.add_option('--mod2obs',dest='mod2obs',default='./mod2obs.exe')
    parser.add_option('--obs2real',dest='obs2real',default='./obs2real.py')
    parser.add_option('--dtrkmf',dest='dtrkmf',default='./dtrkmf.exe')
    return parser


#############################################################
#  ParameterOptions Class Definitions
INIT2PEST: The object-oriented methods of Python allow for classes which contain both data and functions. This class contains the parameter
data for a specific type of parameter (T, for example), and then the commands are used to create the appropriate files.
class ParameterOptions:
    gridFile = None
    coordFile = None
    nZones = 1
```

```
    znInptIndFile = []
    znInptIndVal = []
    outZoneNumFile = None
    spVarFile = None
    znInitValEqn = []
    outInitFieldFile = None
    fpFile = None
    srchDist = 500.0
    znSrchDist = []
    gridDist = 5000.0
    znGridDist = []
    znNoFixed = []
    znNoWells = []
    znNoGrid = []
    wellFile = None
    outTplFile = None
    outPptFile = None
    outR2MFile = None
    outP2FFile = None
    outF2RFile = None
    outFactorsFile = None
    outKrigFieldFile = None
    outModflowFile = None
    maskVal = -999
    varioFile = None
    znVarioModel = []
    znVarioSrch = []
    maxValue = 10.0
    minValue = -10.0
    znMaxValue = []
    znMinValue = []
    varioModel = None
    varioSrch = 0.0
    transform = None
    scaleUp = None
    scaleDown = None
    # Now, non-parsed options
    zoneField = None
    initField = None
    spvarField = None
    pointLocs = None
    pointGrps = None
    wellCoords = {}
    specs = ( 0 , 1 , 1 , 0.0 , 0.0 , 0.0 , 1.0 , 1.0 , 1.0 )
    param = ""
    needMaskParam = False
    parData = []
    parGrps = []
    parFiles = []
    latexTableInfo = None
    latexPointInfo = None

    def __init__( self, param="" , options=None ):
        self.parFiles = []
        self.parData = []
        self.parGrps = []
        self.load_options(options)
        self.param = param
```

INIT2PEST:This section loads the values from the "spec_points_T.in" file, for example, listed in the "spec_config".in file.

```
    def load_options( self, options , baseOpt=None ):
        if options is None: return
        if baseOpt is not None:
            case = baseOpt.peststub
        else:
            case = 'culebra'

        if options.gridFile is not None:
            self.gridFile = options.gridFile%{'case': case, 'param': self.param}
            self.specs = read_spc_file(self.gridFile)

        if options.coordFile is not None:
            self.coordFile = options.coordFile%{'case': case, 'param': self.param}
            self.wellCoords,dummy = read_crd_file(self.coordFile)
        elif baseOpt is not None and baseOpt.coordFile is not None:
            self.coordFile = baseOpt.coordFile%{'case': case, 'param': self.param}
            self.wellCoords,dummy = read_crd_file(self.coordFile)
```

```
        if options.outZoneNumFile is not None:
            self.outZoneNumFile = options.outZoneNumFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.zonefmt is not None:
            self.outZoneNumFile = baseOpt.zonefmt%{'case': case, 'param': self.param}
        else:
            self.outZoneNumFile = 'modeled_zones_points_'+self.param+'.inf'

        if options.spVarFile is not None:
            self.spVarFile = options.spVarFile%{'case': case, 'param': self.param}

        if options.outInitFieldFile is not None:
            self.outInitFieldFile = options.outInitFieldFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.initfmt is not None:
            self.outInitFieldFile = baseOpt.initfmt%{'case': case, 'param': self.param}
        else:
            self.outInitFieldFile = 'modeled_init_points_'+self.param+'.mod'

        if options.fpFile is not None:
            self.fpFile = options.fpFile%{'case': case, 'param': self.param}

        if options.srchDist is not None: self.srchDist = options.srchDist

        if options.gridDist is not None: self.gridDist = options.gridDist

        if options.wellFile is not None: self.wellFile = options.wellFile%{'case': case, 'param':
self.param}

        if options.outPptFile is not None:
            self.outPptFile = options.outPptFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.ppffmt is not None:
            self.outPptFile = baseOpt.ppffmt%{'case': case, 'param': self.param}
        else: self.outPptFile = 'modeled_points_'+self.param+'.dat'

        if options.outTplFile is not None:
            self.outTplFile = options.outTplFile%{'case': case, 'param': self.param}
        else:
            pptpl = self.outPptFile.split('.')
            pptpl[-1] = 'tpl'
            self.outTplFile = '.'.join(pptpl)

        if options.outR2MFile is not None:
            self.outR2MFile = options.outR2MFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.inpredfmt is not None:
            self.outR2MFile = baseOpt.inpredfmt%{'case': case, 'param': self.param, 'prog':
'real2mod'}

        if options.outP2FFile is not None:
            self.outP2FFile = options.outP2FFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.inpredfmt is not None:
            self.outP2FFile = baseOpt.inpredfmt%{'case': case, 'param': self.param, 'prog':
'ppk2fac'}

        if options.outF2RFile is not None:
            self.outF2RFile = options.outF2RFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.inpredfmt is not None:
            self.outF2RFile = baseOpt.inpredfmt%{'case': case, 'param': self.param, 'prog':
'fac2real'}

        if options.outFactorsFile is not None:
            self.outFactorsFile = options.outFactorsFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.facfmt is not None:
            self.outFactorsFile = baseOpt.facfmt%{'case': case, 'param': self.param}
        else:
            self.outFactorsFile = 'modeled_factors_points_'+self.param+'.bin'

        if options.outKrigFieldFile is not None:
            self.outKrigFieldFile = options.outKrigFieldFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.krigfmt is not None:
            self.outKrigFieldFile = baseOpt.krigfmt%{'case': case, 'param': self.param}
        else:
            self.outKrigFieldFile = 'modeled_points_'+self.param+'.mod'

        if options.outModflowFile is not None:
            self.outModflowFile = options.outModflowFile%{'case': case, 'param': self.param}
        elif baseOpt is not None and baseOpt.modfmt is not None:
            self.outModflowFile = baseOpt.modfmt%{'case': case, 'param': self.param}
        else:
            self.outModflowFile = 'modeled_'+self.param+'_field.mod'
```

```
        if options.maskVal is not None: self.maskVal = options.maskVal

        if options.varioFile is not None:
            self.varioFile = options.varioFile%{'case': case, 'param': self.param}

        if options.varioModel is not None: self.varioModel = options.varioModel

        if options.varioSrch is not None: self.varioSrch = options.varioSrch

        if options.transform is not None: self.transform = options.transform

        if options.scaleUp is not None: self.scaleUp = options.scaleUp

        if options.scaleDown is not None: self.scaleDown = options.scaleDown

        if options.maxValue is not None: self.maxValue = options.maxValue
        if options.minValue is not None: self.minValue = options.minValue

        if options.nZones is not None:
            self.nZones = options.nZones
            self.znInptIndFile = [None]*self.nZones
            self.znInptIndVal = [1]*self.nZones
            self.znInitValEqn = [(1.0,0.0,)]*self.nZones
            self.znSrchDist = [self.srchDist]*self.nZones
            self.znGridDist = [self.gridDist]*self.nZones
            self.znNoFixed = [False]*self.nZones
            self.znNoWells = [False]*self.nZones
            self.znNoGrid = [False]*self.nZones
            self.znVarioModel = [self.varioModel]*self.nZones
            self.znVarioSrch = [self.varioSrch]*self.nZones
            self.znMaxValue = [self.maxValue]*self.nZones
            self.znMinValue = [self.minValue]*self.nZones

        if options.znMaxValue is not None:
            for (zn,vl) in options.znMaxValue:
                self.znMaxValue[int(zn)] = float(vl)
        if options.znMinValue is not None:
            for (zn,vl) in options.znMinValue:
                self.znMinValue[int(zn)] = float(vl)

        if options.znInptIndFile is not None:
            for (zn,fl) in options.znInptIndFile:
                if self.znInptIndFile[int(zn)] is None:
                    self.znInptIndFile[int(zn)] = [fl%{'case': case, 'param': self.param}]
                else:
                    self.znInptIndFile[int(zn)].append(fl%{'case': case, 'param': self.param})

        if options.znInptIndVal is not None:
            for (zn,ind) in options.znInptIndVal: self.znInptIndVal[int(zn)] = int(ind)

        if options.znInitValEqn is not None:
            for (zn,m,b) in options.znInitValEqn: self.znInitValEqn[int(zn)] = (float(m),
float(b), )

        if options.znSrchDist is not None:
            for (zn,dist) in options.znSrchDist: self.znSrchDist[int(zn)] = float(dist)

        if options.znGridDist is not None:
            for (zn,dist) in options.znGridDist: self.znGridDist[int(zn)] = float(dist)

        if options.znNoFixed is not None:
            for zn in options.znNoFixed: self.znNoFixed[zn] = True

        if options.znNoWells is not None:
            for zn in options.znNoWells: self.znNoWells[zn] = True

        if options.znNoGrid is not None:
            for zn in options.znNoGrid: self.znNoGrid[zn] = True

        if options.znVarioModel is not None:
            for (zn,md) in options.znVarioModel: self.znVarioModel[int(zn)] = md

        if options.znVarioSrch is not None:
            for (zn,dist) in options.znVarioSrch: self.znVarioSrch[int(zn)] = float(dist)

        return
```

```
    def create_zones( self , baseOpt=None ):
        self.zoneField = FieldData(self.specs)
        self.zoneField.init_to_zeros()
        nZones = self.nZones
        print '  ',nZones,'zones defined for parameter',self.param
        for i in range(1,nZones):
            infile = self.znInptIndFile[i]
            for IFl in infile:
                indic = self.znInptIndVal[i]
                print '    ','reading',IFl,'==',indic,'for zone id ',i
                zD = FieldData(self.specs)
                zD.load_field(IFl,'GEOEAS')
                num = self.zoneField.assign_vals_by_indicator(i,zD,indic)
        zoneFile = self.outZoneNumFile
        self.zoneField.save_field(zoneFile)
        print '    ','saved zone field to:',zoneFile
        return self
```

INIT2PEST: This function creates the initial field from the parameter data, depth, and regression variables. The resulting file can be viewed graphically or textually for verification purposes.

```
    def create_initial_field( self , baseOpt=None ):
        self.initField = FieldData(self.specs)
        self.initField.init_to_zeros()
        varDataFile = self.spVarFile
        SumInfo = [tuple()]*self.nZones
        print '   Creating initial values field for parameter',self.param
        if varDataFile is not None:
            SpatialVar = FieldData(self.specs)
            SpatialVar.load_field(varDataFile,'GEOEAS')
        else:
            SpatialVar = None
        for i in range(0,self.nZones):
            (m,b) = self.znInitValEqn[i]
            (ct,ave) = self.zoneField.assign_initial_by_zone(i,SpatialVar,m,b)
            if SpatialVar is None: D = None
            else: D = 'D'
            SumInfo[i] = ( m , D , b , ct , ave , )
            print '    Zone',i,'assigned',ct,'cells with an average value of',ave
        self.initField.d_data = self.zoneField.d_mask
        self.initField.save_field(self.outInitFieldFile)
        self.spvarField = SpatialVar
        print '    saved initial values field to:',self.outInitFieldFile
        self.latexTableInfo = SumInfo
        return self
```

INIT2PEST: This section creates the pilot point locations for a parameter given the options specified in the "spec_*.in" file. This process places fixed pilot points, then pilot points between pumping and observation wells, and then fills in the remainder with a grid, as is described in Section 3.2.The verification is done both graphically by examining pictures such as in Figure 3-7, but also in console output as presented in H.4.

```
    def create_pilot_points( self , baseOpt=None ):
        (nx,ny,nz,x0,y0,z0,dx,dy,dz) = self.specs
        zoneFld = self.zoneField
        param = self.param
        nZones = self.nZones
        fpFile = self.fpFile
        wlFile = self.wellFile
        PilotPointData = []
        PilotPointGrps = []
        b_noFixed = self.znNoFixed
        b_noWells = self.znNoWells
        b_noGrid = self.znNoGrid
        dist = self.znSrchDist
        gdist = self.znGridDist
        fpVals = {}
        print '   Creating pilot points for parameter',self.param
        if fpFile is not "" and fpFile is not None:
            FixedPoints = read_pts_list(fpFile)
            FixedPointsVals = read_pts_vals(fpFile)
            for (x,y,zn,val) in FixedPointsVals:
                fpId = '%7d%7d'%(x,y,)
                fpVals[fpId] = val
        else:
            FixedPoints = []
        if wlFile is not "" and wlFile is not None:
            WellLocs = read_well_pairs(wlFile)
        else: WellLocs = {}
        b_needMask = False
        PntInfo = [tuple()]*self.nZones
```

```
        for i in range(0,nZones):
            znWellPoints = []
            znGridPoints = []
            ZoneFixed = []
            pointsFld = FieldData(zoneFld.specs)
            pointsFld.init_to_zeros()
            if not b_noFixed[i]:
                pointsFld.blank_fixed_points(FixedPoints,dist[i])
            if not b_noWells[i]:
                znWellPoints = pointsFld.add_well_points(WellLocs,self.wellCoords,dist[i])  ####
Fix WellLocs
            if not b_noGrid[i]:
                znGridPoints = pointsFld.add_grid_points(gdist[i])
            pointsLocations = znWellPoints + znGridPoints
#            print '    Number of possible locations:',len(pointsLocations)
            ZonePoints = self.zoneField.assign_ppoints_to_zone(i,pointsLocations,dist[i])
#            print '    Locations used in zone',i,':',len(ZonePoints)
            vCt = 0
            vMin = self.znMinValue[i]
            vMax = self.znMaxValue[i]
            (m,b) = self.znInitValEqn[i]
            for (x,y,zn) in ZonePoints:
                if self.spvarField is None: D = 0
                else:
                    k = int(( x - x0 ) / dx)
                    j = int(( y0 - y ) / dy)
                    node = k + ( j * nx )
                    D = self.spvarField.d_data[node]
                val = m * D + b
                pId = '%s_z%.1dp%.4d' % ( param, zn, vCt,)
                vCt = vCt + 1
                PilotPointData.append((pId,x,y,zn,val,vMin,vMax,False,self.param+"_zn%d"%i,))
            if not b_noFixed[i]:
                vCt = 0
                ZoneFixed = self.zoneField.assign_ppoints_to_zone(i,FixedPoints,0.00)
                for (x,y,zn) in ZoneFixed:
                    fpId = '%7d%7d'%(x,y,)
                    val = fpVals[fpId]
                    pId = '%s_z%.1df%.4d' % ( param, zn, vCt,)
                    vCt = vCt + 1
                    PilotPointData.append((pId,x,y,zn,val,val,val,True,self.param+"_zn%d"%i,))
            if len(ZonePoints)+len(ZoneFixed) > 0:
                PilotPointGrps.append((self.param+"_zn%d"%i,max(((vMax-vMin)/64.0),0.01),))
            else:
                self.needMaskParam = True
            print '    Assigned',len(ZonePoints),'variable and ',len(ZoneFixed),'fixed pilot
points to zone',i
            PntInfo[i] = ( len(ZonePoints) , len(ZoneFixed) , vMin , vMax , )
        write_ppt_tpl_file(self.outTplFile,PilotPointData)
        print '    saved pilot points template file to:',self.outTplFile
        write_ppt_init_file(self.outPptFile,PilotPointData)
        print '    saved initial pilot points data to:',self.outPptFile
        self.parData = PilotPointData
        self.parGrps = PilotPointGrps
        self.latexPointInfo = PntInfo
        return
```

COMMENT:This section is used, the the resulting file is unused.

```
    def print_latex_table(self ):
        latexFile = self.outInitFieldFile.split('.')
        latexFile[-1] = 'tex'
        latexFile = '.'.join(latexFile)
        LTX = open(latexFile,'w')
        print >>LTX, '\\begin{tabular}{clccccc}'
        print >>LTX, '\\bf Zone &                                                    & \\bf Num   & \\bf
Mean  & \\bf Fixed  & \\bf Variable & \\bf Point Limits \\\\ '
        print >>LTX, '\\bf ID   & \\bf Equation for Initial $\\log_{10}(%s)$ & \\bf Cells & \\bf
Value & \\bf Points & \\bf Points   & $[LB,UB]$ \\\\\\hline '%(self.param)
        for i in range(0,self.nZones):
            ( m , D , b , ct , ave , ) = self.latexTableInfo[i]
            ( nVar , nFix , lb , ub , ) = self.latexPointInfo[i]
            if D is None:
                print >>LTX, 'Z-%d & $%s_{%d} = %f $ & %d & %g & %d & %d & $[%.1f,%.1f]$
\\\\'%(i,self.param,i,b,ct,ave,nFix,nVar,lb,ub)
            else:
                print >>LTX, 'Z-%d & $%s_{%d} = %f \\cdot %s %+.4f $ & %d & %.3f & %d & %d &
$[%.1f,%.1f]$ \\\\'%(i,self.param,i,m,D,b,ct,ave,nFix,nVar,lb,ub)
        print >>LTX, '\\hline\\hline\\end{tabular}'
```

```
        LTX.close()
        print '    saved initial pilot points summary to:',latexFile
        return

    INIT2PEST: This section creates the template files for PPK2FAC, FAC2REAL, and REAL2MOD. The output files can be visually inspected,
    though any failures are quickly evidenced in any log file.
    def create_utility_inputs( self , baseOpt=None ):
        print '   Creating groundwater-utility program input-redirection files'
        opts = { 'gridspec': self.gridFile,
                 'input': self.outKrigFieldFile,
                 'output': self.outModflowFile }
        if self.transform is not None: opts['transform'] = self.transform
        if self.scaleUp is not None: opts['scale-up'] = self.scaleUp
        if self.scaleDown is not None: opts['scale-down'] = self.scaleDown
        self.parFiles.append( ( self.outTplFile, self.outPptFile, ) )
        if self.needMaskParam:
            self.parGrps.append((self.param+"_msk",0.05,))
            self.parData.append(('%s_shift'%self.param,0,0,0,0,-5,5,False,self.param+"_msk",))
            outTplReal2Mod = self.outR2MFile.split('.')
            outTplReal2Mod[-1] = 'tpl'
            outTplReal2Mod = '.'.join(outTplReal2Mod)
            self.parFiles.append( ( outTplReal2Mod , self.outR2MFile , ) )
            opts['mask-value'] = self.maskVal
            opts['mask-shift'] = '#%s_shift    #'%self.param
            opts['mask-file'] = self.outInitFieldFile
            write_r2m_tpl_file(outTplReal2Mod,opts)
            print '    saved REAL2MOD template file to:',outTplReal2Mod
            opts['mask-shift'] = 0.0
        write_r2m_opt_file(self.outR2MFile,opts)
        print '    saved REAL2MOD input file to:',self.outR2MFile
        fac2realInp = []
        fac2realInp.append(self.outFactorsFile)
        fac2realInp.append('u')
        fac2realInp.append(self.outPptFile)
        fac2realInp.append('s')
        fac2realInp.append(-99.9)
        fac2realInp.append('s')
        fac2realInp.append(99.9)
        fac2realInp.append(self.outKrigFieldFile)
        fac2realInp.append('f')
        fac2realInp.append(self.maskVal)
        write_txt_file(self.outF2RFile,fac2realInp)
        print '    saved FAC2REAL input file to:',self.outF2RFile
        ppk2facInp = []
        ppk2facInp.append(self.gridFile)
        ppk2facInp.append(self.outPptFile)
        ppk2facInp.append('1')
        ppk2facInp.append(self.outZoneNumFile)
        ppk2facInp.append(self.varioFile)
        for i in range(0,self.nZones):
            ppk2facInp.append(self.znVarioModel[i])
            ppk2facInp.append('o')
            ppk2facInp.append(self.znVarioSrch[i])
            ppk2facInp.append(1)
            ppk2facInp.append(9)
        ppk2facInp.append(self.outFactorsFile)
        ppk2facInp.append('u')
        ppk2facInp.append('/dev/null')
        ppk2facInp.append('u')
        ppk2facInp.append('/dev/null')
        write_txt_file(self.outP2FFile,ppk2facInp)
        print '    saved PPK2FAC input file to:',self.outP2FFile
        return

    COMMENT:Returns list of data to an external function
    def get_pest_parameters( self , baseOpt=None ):
        return self.parData, self.parGrps, self.parFiles

    INIT2PEST:Parses the "spec_points_*.in" files.
    def option_parser( self , filename=None , baseOpt=None ):
        parser = OptionParser()
        parser.add_option("--grid-specification-file",dest="gridFile", type="string")
        parser.add_option("--well-coordinates-file",dest="coordFile", type="string")
        parser.add_option("--number-of-zones",dest="nZones", type="int")
        parser.add_option("--output-zone-file",dest="outZoneNumFile", type="string")
        parser.add_option("--spatial-variable-file",dest="spVarFile", type="string")
        parser.add_option("--output-initial-value-field",dest="outInitFieldFile", type="string")
```

```
        parser.add_option("--fixed-points-data-file",dest="fpFile", type="string")
        parser.add_option("--search-distance",dest="srchDist", type="float", default=500.0)
        parser.add_option("--pilot-point-grid-spacing",dest="gridDist", type="float")
        parser.add_option("--no-fixed-in-zone",dest="znNoFixed", type="int", action="append")
        parser.add_option("--no-wells-in-zone",dest="znNoWells", type="int", action="append")
        parser.add_option("--no-grids-in-zone",dest="znNoGrid", type="int", action="append")
        parser.add_option("--connect-wells-file",dest="wellFile", type="string")
        parser.add_option("--output-pilot-point-tpl-file",dest="outTplFile", type="string")
        parser.add_option("--output-pilot-point-file",dest="outPptFile", type="string")
        parser.add_option("--real2mod-file",dest="outR2MFile", type="string")
        parser.add_option("--ppk2fac-file",dest="outP2FFile", type="string")
        parser.add_option("--fac2real-file",dest="outF2RFile", type="string")
        parser.add_option("--kriging-factors-file",dest="outFactorsFile", type="string")
        parser.add_option("--kriged-values-file",dest="outKrigFieldFile", type="string")
        parser.add_option("--modflow-input-file",dest="outModflowFile", type="string")
        parser.add_option("--nan-value-mask",dest="maskVal", type="int", default='-999')
        parser.add_option("--kriging-structure-file",dest="varioFile", type="string")
        parser.add_option("--variogram-model",dest="varioModel", type="string")
        parser.add_option("--variogram-search",dest="varioSrch", type="float")
        parser.add_option("--max-value",dest="maxValue",type="float")
        parser.add_option("--min-value",dest="minValue",type="float")
        parser.add_option("--zone-indicator-file",dest="znInptIndFile", type="string", nargs=2,
action="append")
        parser.add_option("--zone-indicator-value",dest="znInptIndVal", type="int", nargs=2,
action="append")
        parser.add_option("--zone-initial-value-eqn",dest="znInitValEqn", type="float", nargs=3,
action="append")
        parser.add_option("--zone-search-distance",dest="znSrchDist", type="float", nargs=2,
action="append")
        parser.add_option("--zone-pilot-point-grid-spacing",dest="znGridDist", type="float",
nargs=2, action="append")
        parser.add_option("--zone-variogram-model",dest="znVarioModel", type="string", nargs=2,
action="append")
        parser.add_option("--zone-variogram-search",dest="znVarioSrch", type="float", nargs=2,
action="append")
        parser.add_option("--zone-min-value",dest="znMinValue", type="float", nargs=2,
action="append")
        parser.add_option("--zone-max-value",dest="znMaxValue", type="float", nargs=2,
action="append")
        parser.add_option("--real2mod-transform",dest="transform", type="string")
        parser.add_option("--real2mod-scale-down",dest="scaleDown", type="float")
        parser.add_option("--real2mod-scale-up",dest="scaleUp", type="float")
        if filename is not None:
            argfile = open(filename, 'r')
            alist = []
            for arg in argfile:
                aa = arg.split()
                for a in aa: alist.append(a.strip())
            (options, args) = parser.parse_args(args=alist)
            self.load_options(options,baseOpt)
        return parser


INIT2PEST: This procedure creates the PEST control file used for the initial PEST execution (r???_init.pst). If this fails, PEST failes and exits
with a non-zero error code. The resulting output file can also be verfied using the PEST utility PESTCHEK.
def build_pest_file( options , fileplus=None , obsData=[] , obsGrps=[] , obsFiles=[] ,
                     parData=[] , parGrps=[] , parFiles=[] , NITER=1):
    if fileplus is not None: fileplus = '_' + fileplus
    else: fileplus = ''
    filename = options.peststub + fileplus + '.pst'
    outfile = open(filename,'w')
    print >>outfile,  'pcf'
    print >>outfile,  '* control data'
    print >>outfile,  'restart estimation'
    print >>outfile,  len(parData),len(obsData),len(parGrps),'0',len(obsGrps)
    print >>outfile,  len(parFiles),len(obsFiles),'single','point',1,0,0
    print >>outfile,  '10.0 2.0 0.10 0.01 1'
    print >>outfile,  '3.0 4.0 0.001'
    print >>outfile,  '0.1'
    print >>outfile,  NITER,'0.001 4 3 0.001 3'
    print >>outfile,  '1 1 1'
    print >>outfile,  '* singular value decomposition'
    print >>outfile,  '1'
    print >>outfile,  int(len(parData)/2),'1e-4'
    print >>outfile,  '1'
    print >>outfile,  '* parameter groups'
    for GRP,DERINC in parGrps:
        print >>outfile,  GRP,'rel_to_max',DERINC,'0.1 switch 2.0 parabolic'
```

```
    print >>outfile,  '* parameter data'
    for PARNME,x,y,zn,PARVAL1,PARLBND,PARUBND,PARTRANS,PARGP in parData:
        if PARTRANS is True: PARTRANS = 'fixed'
        else: PARTRANS = 'none'
        print >>outfile,
PARNME,PARTRANS,'relative',PARVAL1+20,PARLBND+20,PARUBND+20,PARGP,'1.0','-20.0','1'
    print >>outfile,  '* observation groups'
    for OBGNME in obsGrps:
        print >>outfile,  OBGNME
    print >>outfile,  '* observation data'
    for OBSNME,OBSVAL,WEIGHT,OBGNME in obsData:
        print >>outfile,  OBSNME,OBSVAL,WEIGHT,OBGNME
    print >>outfile,  '* model command line'
    print >>outfile, options.model
    print >>outfile,  '* model input/output'
    for TEMPFLE,INFLE in parFiles:
        print >>outfile,  TEMPFLE,INFLE
    for INSFLE,OUTFLE in obsFiles:
        print >>outfile,  INSFLE,OUTFLE
    outfile.close()
    print '  saved PEST control file to:',filename
    return

INIT2PEST: This creates the parallel run control file (r???_init.rmf). Verified by visual inspection once.
def build_pest_rmf(options, fileplus=None):
    if fileplus is not None: fileplus = '_' + fileplus
    else: fileplus = ''
    filename = options.peststub + fileplus + '.rmf'
    outfile = open(filename,'w')
    print >>outfile, 'prf'
    print >>outfile, options.nslaves,'0 2.1000 1'
    for i in range(1,options.nslaves+1):
        print >>outfile, 'slave'+str(i),'./slave'+str(i)
    for i in range(1,options.nslaves+1):
        print >>outfile, '100'
    outfile.close()
    print '  saved PEST run management file:',filename
    return

INIT2PEST: These functions create the zones "modeled_zones_points_?.inf" and place the pilot points in the appropriate manner based on the
options specified in the ParameterData class.
REAL2MOD: Loads a file and log-transforms the inputs
##################################
# FieldData class
class FieldData:
    d_data = []
    d_mask = []
    specs = None
    doMask = None

    def __init__( self , specs=None ):
        if specs is not None:
            self.specs = specs
        else:
            self.specs = ( 0 , 1 , 1 , 0.0, 0.0, 0.0, 1.0, 1.0, 1.0 )
        return

    def init_to_zeros(self):
        nData = self.specs[0] * self.specs[1]
        self.d_data = [0]*nData
        self.d_mask = [0]*nData
        return

    def load_specs(self, filename):
        specs = read_spc_file(filename)
        self.specs = specs
        return

INIT2PEST
    def add_well_points(self, points, coords, distance):
        newPoints = []
        (nx,ny,nz,x0,y0,z0,dx,dy,dz) = self.specs
        ptId = 0
        for (pt1,ptList) in points.iteritems():
            (AX,AY,az) = coords[pt1]
            i = int(( AX - x0 ) / dx)
            j = int(( y0 - AY ) / dy)
```

```
            node = i + ( j * nx )
            if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                newPoints.append((AX,AY,))
                self.d_data[node] = 1
                self.blank_fixed_points([(AX,AY,)],distance*0.8)
            for pt2 in ptList:
                (BX,BY,bz) = coords[pt2]
                Rise = AY-BY
                Run  = AX-BX
                distAB = math.sqrt( (AX-BX)**2 + (AY-BY)**2 )
                NX = AX + (AX-BX)/2.0
                NY = AY + (AY-BY)/2.0
                i = int(( NX - x0 ) / dx)
                j = int(( y0 - NY ) / dy)
                node = i + ( j * nx )
                if distAB < distance*3:
                    if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                        newPoints.append((NX,NY,))
                        self.d_data[node] = 1
                        self.blank_fixed_points([(NX,NY,)],distance*0.8)
                else:
                    Rise = AY-BY
                    Run  = AX-BX
                    distAB = math.sqrt( Run**2 + Rise**2 )
                    if Rise == 0:
                        ddx = distance
                        ddy = 0.0
                    elif Run == 0:
                        ddx = 0.0
                        ddy = distance
                    else:
                        ddx = distance * Run / distAB
                        ddy = distance * Rise / distAB
                    NX1 = AX - ddx
                    NY1 = AY - ddy
                    i = int(( NX1 - x0 ) / dx)
                    j = int(( y0 - NY1 ) / dy)
                    node = i + ( j * nx )
                    if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                        newPoints.append((NX1,NY1,))
                        self.d_data[node] = 1
                        self.blank_fixed_points([(NX1,NY1,)],distance*0.8)
                    NX2 = BX + ddx
                    NY2 = BY + ddy
                    i = int(( NX2 - x0 ) / dx)
                    j = int(( y0 - NY2 ) / dy)
                    node = i + ( j * nx )
                    if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                        newPoints.append((NX2,NY2,))
                        self.d_data[node] = 1
                        self.blank_fixed_points([(NX2,NY2,)],distance*0.8)
        distance = distance * 1.5
        for (pt1,ptList) in points.iteritems():
            (AX,AY,az) = coords[pt1]
            for pt2 in ptList:
                (BX,BY,bz) = coords[pt2]
                Rise = AY-BY
                Run  = AX-BX
                distAB = math.sqrt( Run**2 + Rise**2 )
                if Rise == 0:
                    ddx = distance
                    ddy = 0.0
                elif Run == 0:
                    ddx = 0.0
                    ddy = distance
                else:
                    ddx = 500.0 * Run / distAB
                    ddy = 500.0 * Rise / distAB
                NX = AX + ddx
                NY = AY + ddy
                i = int(( NX - x0 ) / dx)
                j = int(( y0 - NY ) / dy)
                node = i + ( j * nx )
                if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                    newPoints.append((NX,NY,))
                    self.d_data[node] = 1
                    self.blank_fixed_points([(NX,NY,)],distance*0.8)
                NX = BX - ddx
```

```
                NY = BY - ddy
                i = int(( NX - x0 ) / dx)
                j = int(( y0 - NY ) / dy)
                node = i + ( j * nx )
                if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                    newPoints.append((NX,NY,))
                    self.d_data[node] = 1
                    self.blank_fixed_points([(NX,NY,)],distance*0.8)
                if distAB >= distance*3:
                    Rise = AY-BY
                    Run  = AX-BX
                    distAB = math.sqrt( Run**2 + Rise**2 )
                    if Rise == 0:
                        ddx = distance
                        ddy = 0.0
                    elif Run == 0:
                        ddx = 0.0
                        ddy = distance
                    else:
                        ddx = distance * Run / distAB
                        ddy = distance * Rise / distAB
                    for k in range(1,int(distAB/distance)):
                        NX1 = AX - ddx*(k+1)
                        NY1 = AY - ddy*(k+1)
                        i = int(( NX1 - x0 ) / dx)
                        j = int(( y0 - NY1 ) / dy)
                        node = i + ( j * nx )
                        if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                            newPoints.append((NX1,NY1,))
                            self.d_data[node] = 1
                            self.blank_fixed_points([(NX1,NY1,)],distance*0.8)
                        NX2 = BX + ddx*(k+1)
                        NY2 = BY + ddy*(k+1)
                        i = int(( NX2 - x0 ) / dx)
                        j = int(( y0 - NY2 ) / dy)
                        node = i + ( j * nx )
                        if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                            newPoints.append((NX2,NY2,))
                            self.d_data[node] = 1
                            self.blank_fixed_points([(NX2,NY2,)],distance*0.8)
        return newPoints

INIT2PEST
    def add_grid_points(self, distance):
        newPoints = []
        (nx,ny,nz,x0,y0,z0,dx,dy,dz) = self.specs
        ptId = 0
        for iCt in range(0,1+nx/int(distance/dx)):
            for jCt in range(0,1+ny/int((math.sqrt(3)/2)*distance/dy)):
                i = iCt * int(distance/dx) + int(0.2*distance/dx) + ((jCt % 2) *
int(0.5*distance/dx))
                j = jCt * int(distance/dy) + int(0.2*distance/dy)
                node = i + ( j * nx )
                x = x0 + i * dx
                y = y0 - j * dy
                if not i < 0 and not j < 0 and node < nx*ny and self.d_mask[node] == 0:
                    newPoints.append((x,y,))
                    self.d_data[node] = 1
                    self.blank_fixed_points([(x,y,)],distance*0.6)
        return newPoints


INIT2PEST
    def blank_fixed_points(self, points, distance):
        # Points are array of (x,y,) tuples
        (nx,ny,nz,x0,y0,z0,dx,dy,dz) = self.specs
        iDist = int(distance/dx)
        jDist = int(distance/dy)
        for (x,y) in points:
            i = int(( x - x0 ) / dx)
            j = int(( y0 - y ) / dy)
            for ii in range(-iDist,iDist+1):
                for jj in range(-jDist,jDist+1):
                    if i + ii < 0 or i + ii >=nx: continue
                    if j + jj < 0 or j + jj >=ny: continue
                    if ii < 0: sii = -1
                    else: sii = 1
                    if jj < 0: sjj = -1
```

```
                    else: sjj = 1
                    xcur = x0 + ( i + ii + (sii*0.5) ) * dx
                    ycur = y0 - ( j + jj + (sjj*0.5) ) * dy
                    dist = math.sqrt( ( x - xcur)**2 + ( y - ycur)**2 )
                    node = i + ii + ( j + jj ) * nx
                    if dist <= distance: self.d_mask[int(node)] = -1
        return
```

INIT2PEST

```
    def assign_vals_by_indicator( self , Value , IndicField, IndicValue=1 ):
        ct = 0
        if len(self.d_data) <> len(IndicField.d_data):
            raise 'fields are not the same size!'
        for i in range(0,len(self.d_data)):
            v = IndicField.d_data[i]
            if v == IndicValue:
                self.d_data[i] = Value
                ct = ct + 1
        return ct
```

INIT2PEST

```
    def assign_initial_by_zone( self, ZoneID, SpatialVar=None, m=1.0 , b=0.0 ):
        ct = 0
        mySum = 0.0
        for i in range(0,len(self.d_data)):
            z = self.d_data[i]
            if SpatialVar is not None:
                D = SpatialVar.d_data[i]
            else: D = 0.0
            if z == ZoneID:
                v = m * D + b
                self.d_mask[i] = v
                mySum = mySum + v
                ct = ct + 1
        return ct, float(mySum/ct)
```

INIT2PEST

```
    def assign_ppoints_to_zone( self, ZoneID, points=[], distance=500.):
        ct = 0
        (nx,ny,nz,x0,y0,z0,dx,dy,dz) = self.specs
        iDist = int(distance/dx)
        jDist = int(distance/dy)
        goodPts = []
        for (x,y) in points:
            i = int(( x - x0 ) / dx)
            j = int(( y0 - y ) / dy)
            node = i + ( j * nx )
            z= self.d_data[node]
            if z == ZoneID:
                goodPts.append( (x,y,z,) )
            else:
                addPt = False
                for ii in range(-iDist,iDist+1):
                    for jj in range(-jDist,jDist+1):
                        if i + ii < 0 or i + ii >=nx: continue
                        if j + jj < 0 or j + jj >=ny: continue
                        node = i + ii + ( j + jj ) * nx
                        z =self.d_data[node]
                        if z == ZoneID: addPt = True
                z = ZoneID
                if addPt:
                    goodPts.append( (x,y,z,) )
        return goodPts
```

REAL2MOD & INIT2PEST

```
    def load_field(self, filename, format="MODFLOW" ):
        fmt = format.upper()
        if fmt == "MODFLOW":
            self.d_data = read_mf2k(filename)
        elif fmt == "GEOEAS":
            self.d_data = read_geoeas(filename,self.specs)
        else:
            print 'cannot open', filename, 'of unknown format', format
            raise
        return
```

REAL2MOD & INIT2PEST

```
    def save_field(self, filename, format="MODFLOW" ):
        fmt = format.upper()
        if fmt == "MODFLOW":
            self.write_mf2k(filename)
        else:
            print 'cannot save', filename, 'of unknown format', format
            raise
        return

REAL2MOD & INIT2PEST
    def load_mask(self, filename, format="MODFLOW" ):
        fmt = format.upper()
        if fmt == "MODFLOW":
            self.d_mask = read_mf2k(filename)
        elif fmt == "GEOEAS":
            self.d.mask = read_geoeas(filename,self.specs)
        else:
            print 'cannot open', filename, 'of unknown format', format
            raise
        return

REAL2MOD
    def run_transforms(self, transforms):
        ct = 0
        ( shift , scale , maskShft , xform ) = transforms
        for val in self.d_data:
            if val == self.doMask: val = self.d_mask[ct] + maskShft
            if xform == "LOG10":
                val = val * scale
                val = math.log10(val)
                val = val + shift
            elif xform == "POW10":
                val = val + shift
                val = 10**val
                val = val * scale
            else:
                val = val * scale
                val = val + shift
            self.d_data[ct] = val
            ct = ct + 1
        return

REAL2MOD & INIT2PEST
    def write_mf2k( self, filename ):
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
        else:
            for val in self.d_data:
                print >> outfile, val
        outfile.close()
        return

INIT2PEST: The following calculates initial pilot point values based on the zone, depth, and regression parameters.
####################################
# PointData class
class PointData:
    PilotPointLocs = []
    FixedPointLocs = []
    FixedPointVals = []
    InitialValPars = []
    InitialValSPVF = None
    need_mask = []
    max_values = []
    min_values = []
    nZones = 1

    def __init__(self, nZones=1):
        self.need_mask = [False]*nZones
        self.max_values = [1]*nZones
        self.min_values = [0]*nZones
        self.nZones = nZones
        return

    def assign_parameters( self , param="" ):
```

```
        parameters = []
        b_useSPV = False
        znPtCt = [0]*self.nZones
        znFpCt = [0]*self.nZones
        if self.InitialValSPVF is None: D = 0.0
        else:
            b_useSPV = True
            spvFld = self.InitialValSPVF
        for (x,y,zn) in self.PilotPointLocs:
            x = float(x)
            y = float(y)
            zn = int(zn)
            if b_useSPV:
                (nx,ny,nz,x0,y0,z0,dx,dy,dz) = spvFld.specs
                i = int(( x - x0 ) / dx)
                j = int(( y0 - y ) / dy)
                node = i + ( j * nx )
                D = spvFld.d_data[node]
            (m,b) = self.InitialValPars[zn]
            iVal = D * m + b
            vMin = self.min_values[zn]
            vMax = self.max_values[zn]
            vCt = znPtCt[zn]
            znPtCt[zn] = vCt + 1
            pId = '%s_z%.1dp%.4d' % ( param, zn, vCt,)
            bIsFixed = False
            bIsMask = False
            parameters.append( (pId, x , y , zn , iVal , vMin , vMax , bIsFixed , bIsMask ,
"grp_"+param, ) )
        fpVals = {}
        for (x,y,zn,val) in self.FixedPointVals:
            fpId = '%7d%7d'%(x,y,)
            fpVals[fpId] = (x , y , val , )
        for (x,y,zn) in self.FixedPointLocs:
            fpId = '%7d%7d'%(x,y,)
            (xx,yy,iVal) = fpVals[fpId]
            bIsFixed = True
            bIsMask = False
            vCt = znFpCt[zn]
            znFpCt[zn] = vCt + 1
            pId = '%s_z%.1df%.4d' % ( param , zn, vCt, )
            parameters.append( (pId, x , y , zn , iVal , iVal , iVal , bIsFixed , bIsMask ,
"grp_"+param,  ) )
        b_mask = False
        for useMask in self.need_mask:
            if useMask: b_mask = True
        if b_mask:
            pId = '%s_shiftzn' % (param,)
            parameters.append( (pId, 0.0, 0.0, 0 , 0.0 , -5.0 , 5.0 , False, True , "msk_"+param,
) )
        return parameters
```

INIT2PEST and OBS2REAL:

```
###################################
# Sample Data class
class SampleData:
    "well sample data (head or drawdown) observations"
    measured = {}
    modeled = {}
    wells = []
    pumpID = None
    well_coords = {}
    residuals = {}
    rmseVals = {}
    areaVals = {}
    sseVals = {}
    aveArea = {}
    nrmseVals = {}
    maxMeas = {}
    maxModl = {}
    b_singleGroup = False
    startdate = None
    def __init__(self, pumpID=None, startDate=None):
        "initialize the SampleData object"
        self.pumpID = pumpID
        if startDate:
            self.startdate = datetime.datetime(*time.strptime(startDate,
                                                    '%m/%d/%Y %H:%M:%S')[0:6])
```

```
COMMENT: Next routines are I/O for reading and writing files specified in the options files
    def print_mod2obs_file(self, filename):
        'SPCFILE'
        'CRDFILE'
        'CRDFILE'
        'MEASFILE'
        'MF2KFILE'
        'f'
        '1999'
        's'
        'STDATE'
        'STTIME'
        '1'
        '365'
        'MODLFILE'
        return

    def print_obs2real_file(self, filename):
        pass

    def load_measured(self, filename):
        "load 'true' field data"
        self.measured , self.wells = read_smp_file(filename)
        return

    def load_modeled(self, filename):
        "load modeled observations"
        self.modeled , self.wells = read_smp_file(filename)
        return

    def load_coordinates(self, filename):
        "load well coordinates"
        self.well_coords , dummy = read_crd_file(filename)
        return

OBS2REAL: calculates the difference in area between to curves.
    def compare_samples(self):
        "calculate error metrics between measured ('true') and modeled results"
        for well in self.wells:
            errorStack = []
            ms = self.measured[well]
            md = self.modeled[well]
            lastDate = self.startdate
            lastMs = lastMd = 0.0
            lastResid = 0.0
            sse = 0.0
            cumArea = 0.0
            cumDelta = 0.0
            maxMeas = 0.0001
            minMeas = 0.0000
            maxModeled = 0.0001
            tmAtMxMs = 1
            tmAtMxMd = 1
            for i in range(0,len(ms)):
                (msTm,msVl) = ms[i]
                (mdTm,mdVl) = md[i]
                if msTm <> mdTm:
                    print 'files do not match!'
                    raise
                if lastDate is None:
                    lastDate = msTm
                diff = msTm - lastDate
                delta = diff.days*60*60*24 + diff.seconds
                if delta == 0.0: delta = 1.0
                cumDelta = cumDelta + delta
                if maxMeas is None or msVl > maxMeas:
                    maxMeas = msVl
                    tmAtMxMs = cumDelta
                if minMeas is None or msVl < minMeas: minMeas = msVl
                if maxModeled is None or mdVl > maxModeled:
                    maxModeled=mdVl
                    tmAtMxMd = cumDelta
                resid = mdVl - msVl
                if resid > 0 > lastResid or resid < 0 < lastResid:
                    slMs = ( msVl - lastMs ) / delta
                    slMd = ( mdVl - lastMd ) / delta
                    midPt = ( lastMs - lastMd ) / (slMd - slMs)
```

```
                    area1 = abs(lastResid) * 0.5 * midPt
                    area2 = abs(resid) * 0.5 * ( delta - midPt )
                    area = area1 + area2
                else:
                    area1 = abs(lastResid) * delta * 0.5
                    area2 = abs(resid) * delta * 0.5
                    area = area1 + area2
                lastDate = msTm
                lastResid = resid
                lastMs = msVl
                lastMd = mdVl
                errorStack.append( (resid, area,) )
                sse = resid**2 + sse
                cumArea = cumArea + area
            self.aveArea[Well] = cumArea / cumDelta
            self.residuals[Well] = errorStack
            self.areaVals[Well] = cumArea
            self.sseVals[Well] = sse
            self.rmseVals[Well] = math.sqrt( sse / len(ms) )
            self.nrmseVals[Well] = math.sqrt( sse / len(ms) ) / ( maxMeas - minMeas )
            if tmAtMxMs < cumDelta / 10: tmAtMxMs = cumDelta
            self.maxMeas[Well] = ( tmAtMxMs, maxMeas,)
            self.maxModl[Well] = ( tmAtMxMd, maxModeled,)
            pass
        return
```

OBS2REAL: Assignes data from input files into PEST formatted observations

```
    def print_outfile(self, filename=None, b_singleGroup=False):
        if b_singleGroup: self.b_singleGroup = True
        "print only the modeled values and graph-error metrics for PEST"
        if filename is None:
            outfile = sys.stdout
        else:
            filename = filename + '.out'
            try:
                outfile = open(filename,'w')
            except IOError:
                print 'cannot open', filename
                raise
        for Well in self.wells:
            md = self.modeled[Well]
            for i in range(0,len(md)):
                (mdTm,mdVl) = md[i]
                print >> outfile, mdVl
            if not b_singleGroup: print >> outfile, self.aveArea[Well]
            (tmMs,msMx) = self.maxMeas[Well]
            (tmMd,mdMx) = self.maxModl[Well]
            tmDiff = ( tmMd - tmMs ) / tmMs
            if not b_singleGroup: print >> outfile, tmDiff
        if filename is not None: outfile.close()
        return
```

INIT2PEST: Assignes data from input values into PEST formatted observations

```
    def assign_obs_groups(self, b_singleGroup=False ):
        groups=[]
        if b_singleGroup:
            groups.append(self.pumpID.upper())
        else:
            for Well in self.wells:
                mdGp = self.pumpID.upper() + Well.upper()
                groups.append(mdGp)
        return groups
```

INIT2PEST: Assignes data from input values into PEST formatted observations

```
    def assign_obs_data(self, b_singleGroup):
        observed=[]
        for Well in self.wells:
            md = self.measured[Well]
            ## TODO:  Need to add in weighting options here
            if b_singleGroup:
                mdGp = self.pumpID.upper()
                mdWt = 2.00
            else:
                mdGp = self.pumpID.upper() + Well.upper()
                mdWt = 1.00
            for i in range(0,len(md)):
                (mdTm,mdVl) = md[i]
                mdId = '%-6.6s_%-6.6s_o%.2d' % ( self.pumpID.lower() , Well.lower() ,i )
```

```
                mdId = mdId.replace(' ','_')
                observed.append( (mdId, mdVl, mdWt, mdGp,))
            geId = '%-6.6s_%-6.6s_err' % ( self.pumpID.lower() , Well.lower() )
            geId = geId.replace(' ','_')
            geVl = 0.00
            geWt = math.sqrt(len(md))
            if not b_singleGroup:
                observed.append( (geId, geVl, geWt, mdGp,))
            geId = '%-6.6s_%-6.6s_tim' % ( self.pumpID.lower() , Well.lower() )
            geId = geId.replace(' ','_')
            geVl = 0.00
            geWt = math.sqrt(len(md)/2)
            if not b_singleGroup:
                observed.append( (geId, geVl, geWt, mdGp,))
        return observed

OBS2REAL: Prints the outputs in a more readable format
    def print_summary_data(self, filename=None, append=False):
        "print sumamry error metrics for each observation well"
        if filename is None:
            outfile = sys.stdout
        else:
            try:
                if append: outfile = open(filename,'a')
                else: outfile = open(filename,'w')
            except IOError:
                print 'cannot open', filename
                raise
        if not append:
            print >> outfile, '# Error Metrics Summary File'
            print >> outfile, '# Test_Name, SSE, RMSE, NRMSE, Err-Area, Ave-Err, Measured_Max,
Modeled_Max, N , MeasMaxDdnAt , ModlMaxDdnAt '
        if self.b_singleGroup:
            pass
        else:
            for Well in self.wells:
                md = self.modeled[Well]
                n = len(md)
                print >> outfile, '%-20s %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e %3d %f
%f'%( '"'+self.pumpID+'-'+Well+'"',
                                    self.sseVals[Well], self.rmseVals[Well],
self.nrmseVals[Well],
                                    self.areaVals[Well], self.aveArea[Well],
                                    self.maxMeas[Well][1], self.maxModl[Well][1], n ,
                                    self.maxMeas[Well][0], self.maxModl[Well][0] )
        if filename is not None: outfile.close()
        return

OBS2REAL: Prints the outputs in a more readable format
    def print_full_data(self, filename=None):
        "print full data for each measurement pair"
        if filename is None:
            outfile = sys.stdout
        else:
            try:
                outfile = open(filename,'w')
            except IOError:
                print 'cannot open', filename
                raise
        print >> outfile, '# Full Output Listing File'
        print >> outfile, '# Point_Name, Date, Time, Meas, Modl, Resid, NormMeas, NormModl,
NormResid'
        for Well in self.wells:
            md = self.modeled[Well]
            ms = self.measured[Well]
            for i in range(0,len(md)):
                (msTm,msVl) = ms[i]
                (mdTm,mdVl) = md[i]
                max = self.maxMeas[Well][1]
                print >> outfile, '%-16s %-20s %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e'
%('"'+self.pumpID+'-'+Well+'"',
                                                        msTm.strftime('%m/%d/%Y %H:%M:%S'),
                                                        msVl,mdVl,mdVl-msVl,
                                                        msVl/max,mdVl/max,
                                                        (mdVl-msVl)/max                    )
            print >> outfile, ''
            print >> outfile, ''
        outfile.close()
```

```
        return

ALL: The following routines are I/O routines used by all the utilities. These routines were verified by comparing the expected output to the output
that the routines created. Further verification comes from the fact that PEST and its utilities are syntax-error intolerant, and would crash (and be
so indicated in the logs) if there were any errors.
########################################
# Generic Read/Write Routines
def write_ins_file(filename, observations):
    if filename is None: outfile = sys.stdout
    else:
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
    print >> outfile, 'pif *'
    for (geId, geVl, geWt, mdGp) in observations:
        print >>outfile, 'l1 !' + geId + '!'
    if filename is not None: outfile.close()
    return

def write_ppt_tpl_file(filename, parameters):
    if filename is None: outfile = sys.stdout
    else:
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
    print >> outfile, 'ptf #'
    for (pId,x,y,zn,v,vmin,vmax,fix,grp) in parameters:
        print >>outfile, '%s  %d  %d  %d  #%-15s#'%(pId,x,y,zn,pId)
    if filename is not None: outfile.close()
    return

def write_ppt_init_file(filename, parameters):
    if filename is None: outfile = sys.stdout
    else:
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
    for (pId,x,y,zn,v,vmin,vmax,fix,grp) in parameters:
        print >>outfile, '%s  %d  %d  %d  %15f'%(pId,x,y,zn,v)
    if filename is not None: outfile.close()
    return

def write_r2m_tpl_file(filename, parameters):
    if filename is None: outfile = sys.stdout
    else:
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
    print >> outfile, 'ptf #'
    for (opt,val) in parameters.iteritems():
        print >>outfile, '--'+opt+'='+str(val)
    if filename is not None: outfile.close()
    return

def write_r2m_opt_file(filename, parameters):
    if filename is None: outfile = sys.stdout
    else:
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
    for (opt,val) in parameters.iteritems():
        print >>outfile, '--'+opt+'='+str(val)
    if filename is not None: outfile.close()
    return

def write_txt_file(filename, lines):
    if filename is None: outfile = sys.stdout
```

```
    else:
        try:
            outfile = open(filename,'w')
        except IOError:
            print 'cannot open', filename
            raise
    for line in lines:
        print >>outfile, line
    if filename is not None: outfile.close()
    return

def read_well_pairs( filename ):
    infile = open(filename,'r')
    adict = {}
    for line in infile:
        (w1,w2) = line.strip().upper().split()
        if w1 in adict:
            adict[w1].append(w2)
        else:
            adict[w1] = [w2]
    infile.close()
    return adict

def read_smp_file(filename):
    "read a PEST groundwater utilities Bore Sample File"
    adict = {}
    wells = []
    ThisWell = None
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    else:
        pass
    for line in infile:
        line = line.strip()
        (Well, Date, Time, Meas) = line.split()
        Well = Well.upper()
        MDate = datetime.datetime(*time.strptime(Date + ' ' + Time,
                                                 '%m/%d/%Y %H:%M:%S')[0:6])
        if ThisWell <> Well:
            data = []
            ThisWell = Well
            wells.append(Well)
        data.append( (MDate, float(Meas),) )
        adict[Well] = data
    infile.close()
    return adict, wells

def read_mf2k( filename ):
    "read a formatted MODFLOW data file"
    ct = 0
    data = []
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    for line in infile:
        vals = str.split(line.strip())
        for val in vals:
            data.append(float(val))
            ct = ct + 1
    infile.close()
    return data

def read_geoeas( filename , inSpecs=None):
    "read a GEO-EAS simplified data file"
    data = []
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    header = infile.readline().strip()
    specs = infile.readline().strip().split()
    colHeader=[]
```

```
    nCols = specs[0]
    del specs[0]
    if len(specs) > 8:
        (nx,ny,nz,x0,y0,z0,dx,dy,dz)=specs[0:9]
        nx = int(nx)
        ny = int(ny)
        nz = int(nz)
        x0 = float(x0)
        y0 = float(y0)
        z0 = float(z0)
        dx = float(dx)
        dy = float(dy)
        dz = float(dz)
        y0 = y0 + (ny-1) * dy
    elif inSpecs is not None:
        (nx,ny,nz,x0,y0,z0,dx,dy,dz)=inSpecs[0:9]
    else:
        (nx,ny,nz,x0,y0,z0,dx,dy,dz)=(0,1,1,0.,0.,0.,1.,1.,1.)
    for i in range(0,int(nCols)):
        colHeader.append(infile.readline().strip())
    node = 0
    nData = nx * ny
    data = [0]*nData
    for line in infile:
        thisLine = line.strip().split()
        x = float(thisLine[0])
        del thisLine[0]
        y = float(thisLine[0])
        del thisLine[0]
        z = float(thisLine[0])
        del thisLine[0]
        vals = float(thisLine[0])
        if nx <> 0:
            i = int(( x - x0 ) / dx)
            j = int(( y0 - y ) / dy)
            node = i + j * nx
            data[node] = vals
        else:
            node = node + 1
            data[node] = vals
    infile.close()
    return data

def read_spc_file(filename):
    "read in a PEST GW Utilities Grid Specification File"
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    (ny,nx)=infile.readline().strip().split()
    (x0,y0,z0)=infile.readline().strip().split()
    (nx,dx) = infile.readline().strip().split('*')
    (ny,dy) = infile.readline().strip().split('*')
    infile.close()
    specs = (int(nx),int(ny),1,float(x0)+float(dx)/2.0,float(y0)-
float(dy)/2.0,float(z0),float(dx),float(dy),1.0)
    return specs

def read_crd_file(filename):
    "read a PEST groundwater utilities Bore Coordinates File"
    adict = {}
    wells = []
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    for line in infile:
        vals = str.split(line.strip())
        wellID = vals[0].upper()
        x = float(vals[1])
        y = float(vals[2])
        z = float(vals[3])
        crd = ( x, y, z, )
        wells.append(wellID)
        adict[wellID] = crd
    infile.close()
```

```
    return adict, wells

def read_pts_list(filename):
    "read a set of points locations (ignoring their names or values)"
    data = []
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    for line in infile:
        vals = str.split(line.strip())
        (x,y) = vals[1:3]
        data.append( (float(x),float(y),) )
    infile.close()
    return data

def read_pts_vals(filename):
    "read a set of points locations (ignoring their names)"
    data = []
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    for line in infile:
        vals = str.split(line.strip())
        (x,y,z,v) = vals[1:5]
        data.append( (float(x),float(y),float(z),float(v),) )
    infile.close()
    return data

def read_pumping( filename=None ):
    "read a PEST groundwater utilities Bore Pumping File"
    try:
        infile = open(filename,'r')
    except IOError:
        print 'cannot open', filename
        raise
    else:
        pass
    lastPump = None
    pumpList = []
    startDates = {}
    for line in infile:
        (pumpID, date1, time1, date2, time2, volume) = line.split()
        pumpID = pumpID.upper()
        if pumpID <> lastPump:
            StartDate = date1 + ' ' + time1
            pumpList.append(pumpID)
            startDates[pumpID] = StartDate
            lastPump = pumpID
    infile.close()
    return startDates, pumpList
```

G.5 Recalibration and Analysis Codes

G.5.1 Listing – CVS://Tfields::Inputs/scripts/fix_snl8.sh

This script fixes all points outside the area of interest (see Figure 4-3) and then recreates the PEST control file.

```
#!/bin/bash

FIELD=$1

X_W=616000
Y_S=3580000
X_E=630000
Y_N=3589000

PARAMS="A T R S"
for P in $PARAMS
  do
  grep -v -i "f"  modeled_final_points_${P}.dat | awk -v X1=$X_W -v Y1=$Y_S -v X2=$X_E -v Y2=$Y_N
'{ if ( X1>$2 || Y1>$3 || X2<$2 || Y2<$3 ) printf("%s\tfix_param\t%f\tremove_prior\n",$1,20+$5);
}'
  grep -i ${P}_shift ${FIELD}_last.pst | awk '{printf("%s\tfix_param\t%s\tremove_prior\n",$1,$4);
}'
done> paramfix_points.in
./parrep.exe ${FIELD}_last.par ${FIELD}_init.pst ${FIELD}_update.pst
./paramfix.exe paramfix_points.in ${FIELD}_update.pst ${FIELD}_init.pst
mv ${FIELD}_new.pst ${FIELD}_init.pst
```

G.5.2 Listing - CVS://Tfields::Inputs/scripts/run_dtrkmf.sh

This script checks out the fields from the Outputs directory, then runs DTRKMF on the budget file. The output files are renamed with the field ID (r???) in the prefix, so they can be stored in a single directory.

```
#!/bin/bash

FILES=`cat $1`
for File in $FILES
  do
  FID=${File##*/}
  cvs -d $CVSROOT co -p Outputs/${File}/modeled_K_field.mod > ${FID}_K_field.mod
  cvs -d $CVSROOT co -p Outputs/${File}/modeled_A_field.mod > ${FID}_A_field.mod
  cvs -d $CVSROOT co -p Outputs/${File}/modeled_S_field.mod > ${FID}_S_field.mod
  cvs -d $CVSROOT co -p Outputs/${File}/modeled_R_field.mod > ${FID}_R_field.mod
  cvs -d $CVSROOT co -p -kb Outputs/${File}/modeled_flow.bud > ${FID}_flow.bud
  ln -f elev_top.mod fort.33
  ln -f elev_bot.mod fort.34
  ./dtrkmf.exe dtrkmf_wipplwb.inp ${FID}_flow.bud ${FID}_travel_LWB.dat debug.out
  ./dtrkmf.exe dtrkmf_domain.inp ${FID}_flow.bud ${FID}_travel_domain.dat debug.out
done
```

G.6 Configuration Files

The configuration files used as input to INIT2PEST are provided here for convenience.

G.6.1 Configuration - CVS://Tfields::Inputs/config/spec_setup.in

```
--observations-options-file=spec_observations.in

--parameter-options-file=T spec_points_T.in
--parameter-options-file=S spec_points_S.in
--parameter-options-file=A spec_points_A.in
--parameter-options-file=R spec_points_R.in

--pilotpoints-filename-format=modeled_points_%(param)s.dat
--factors-filename-format=modeled_factors_points_%(param)s.bin
--initial-values-field-filename-format=modeled_init_points_%(param)s.mod
--kriged-parameter-field-filename-format=modeled_points_%(param)s.mod
--final-parameter-field-filename-format=modeled_%(param)s_field.mod
--flow-field-filename-format=modeled_flow.bin
--modflow-output-filename-format=modeled_%(test)s.bin
--modeled-samples-filename-format=modeled_%(test)s.smp
--measured-samples-filename-format=meas_%(test)s.smp
--command-redirect-filename-format=%(prog)s_points_%(param)s.in

--num-slaves=6
--model-command=./model.sh
```

G.6.2 Configuration - CVS://Tfields::Inputs/config/spec_observations.in

```
--steady-state-test=head
--pumping-test-file=spec_pumping.dat
```

G.6.3 Configuration – CVS://Tfields::Inputs/config/spec_points_T.in

```
--grid-specification-file=spec_domain.spc
--well-coordinates-file=spec_locs.crd

--number-of-zones=5
--spatial-variable-file=elev_overburden.geo
--fixed-points-data-file=fixed_points_T.ppt
--search-distance=500.0
--pilot-point-grid-spacing=2000.0
--connect-wells-file=spec_connect_T.lst
--max-value=-1
--min-value=-19

--zone-initial-value-eqn=0 -3.48357e-3 -3.63224

--zone-indicator-file=1 %(case)scoord.map
--zone-indicator-file=1 zone_halitemargins.geo
--zone-initial-value-eqn=1 -3.48357e-3 -5.69805

--zone-indicator-file=2 zone_dissolution.geo
--zone-initial-value-eqn=2 -3.48357e-3 -2.94635

--zone-indicator-file=3 zone_halite.geo
--zone-initial-value-eqn=3 -3.48357e-3 -10.449
--no-fixed-in-zone=3
--no-wells-in-zone=3
--no-grids-in-zone=3

--zone-indicator-file=4 zone_noflow.geo
--zone-initial-value-eqn=4 0.00 -25
--no-fixed-in-zone=4
--no-wells-in-zone=4
--no-grids-in-zone=4

--output-zone-file=modeled_zones_points_T.inf
--output-initial-value-field=modeled_init_points_T.mod
--output-pilot-point-tpl-file=modeled_points_T.tpl
--output-pilot-point-file=modeled_points_T.dat

--ppk2fac-file=ppk2fac_points_T.in
--kriging-factors-file=modeled_factors_points_T.bin
--kriging-structure-file=spec_variogram.str
--variogram-model=culebra
--variogram-search=9500.

--fac2real-file=fac2real_points_T.in
--kriged-values-file=modeled_points_T.mod
--nan-value-mask=-999

--real2mod-file=real2mod_points_T.in
--real2mod-scale-down=7.75
--real2mod-transform=pow10
--modflow-input-file=modeled_K_field.mod
```

G.6.4 Configuration - CVS://Tfields::Inputs/config/spec_points_S.in

```
--grid-specification-file=spec_domain.spc
--well-coordinates-file=spec_wells.crd

--number-of-zones=5
--search-distance=500.0
--fixed-points-data-file=fixed_points_S.ppt
--pilot-point-grid-spacing=2000.0
--connect-wells-file=spec_connect_S.lst
--max-value=-0.5
--min-value=-6

--zone-initial-value-eqn=0 0 -5.0
--zone-max-value=0 -4.5
--zone-min-value=0 -5.5
--no-grids-in-zone=0

--zone-indicator-file=1 zone_transition.geo
--zone-initial-value-eqn=1 0 -4.0
--no-wells-in-zone=1
--no-fixed-in-zone=1
--zone-max-value=1 -0.5
--zone-min-value=1 -6

--zone-indicator-file=2 zone_unconfined.geo
--zone-initial-value-eqn=2 0 -1.5
--zone-max-value=2 -0.5
--zone-min-value=2 -2.5
--no-wells-in-zone=2
--no-fixed-in-zone=2

--zone-indicator-file=3 zone_halite.geo
--zone-initial-value-eqn=3 0 -5.0
--no-fixed-in-zone=3
--no-wells-in-zone=3
--no-grids-in-zone=3

--zone-indicator-file=4 zone_noflow.geo
--zone-indicator-value=4 1
--zone-initial-value-eqn=4 0 -25.0
--no-fixed-in-zone=4
--no-wells-in-zone=4
--no-grids-in-zone=4

--kriging-structure-file=spec_variogram.str
--variogram-model=culebra
--variogram-search=9000.

--nan-value-mask=-999

--real2mod-scale-down=7.75
--real2mod-transform=pow10
```

G.6.5 Configuration – CVS://Tfields::Inputs/config/spec_points_A.in

```
--grid-specification-file=spec_domain.spc
--well-coordinates-file=spec_locs.crd

--number-of-zones=5
--search-distance=750.0
--pilot-point-grid-spacing=3000.0
--fixed-points-data-file=fixed_points_A.ppt
--connect-wells-file=spec_connect_A.lst
--max-value=0.5
--min-value=-0.5

--zone-initial-value-eqn=0 0 0.0
--no-fixed-in-zone=0

--zone-indicator-file=1 %(case)scoord.map
--zone-indicator-file=1 zone_halitemargins.geo
--zone-indicator-value=1 1
--zone-initial-value-eqn=1 0 0.0
--no-fixed-in-zone=1

--zone-indicator-file=2 zone_dissolution.geo
--zone-indicator-value=2 1
--zone-initial-value-eqn=2 0 0.0
--no-fixed-in-zone=2
--no-wells-in-zone=2

--zone-indicator-file=3 zone_halite.geo
--zone-indicator-value=3 1
--zone-initial-value-eqn=3 0 0.0
--no-fixed-in-zone=3
--no-wells-in-zone=3
--no-grids-in-zone=3

--zone-indicator-file=4 zone_noflow.geo
--zone-indicator-value=4 1
--zone-initial-value-eqn=4 0 -25.0
--no-fixed-in-zone=4
--no-grids-in-zone=4
--no-wells-in-zone=4

--output-zone-file=modeled_zones_points_A.inf
--output-initial-value-field=modeled_init_points_A.mod
--output-pilot-point-tpl-file=modeled_points_A.tpl
--output-pilot-point-file=modeled_points_A.dat

--ppk2fac-file=ppk2fac_points_A.in
--kriging-factors-file=modeled_factors_points_A.bin
--kriging-structure-file=spec_variogram.str
--variogram-model=culebra
--variogram-search=5000

--fac2real-file=fac2real_points_A.in
--kriged-values-file=modeled_points_A.mod
--nan-value-mask=-999

--real2mod-transform=pow10
--real2mod-file=real2mod_points_A.in
--modflow-input-file=modeled_A_field.mod
```

G.6.6 Configuration – CVS://Tfields::Inputs/config/spec_points_R.in

```
--grid-specification-file=spec_domain.spc
--well-coordinates-file=spec_recharge.crd
--number-of-zones=2
--output-zone-file=modeled_zones_points_R.inf
--fixed-points-data-file=fixed_points_R.ppt
--output-initial-value-field=modeled_init_points_R.mod
--search-distance=4000.0
--pilot-point-grid-spacing=7000.0
--connect-wells-file=spec_connect_R.lst
--output-pilot-point-tpl-file=modeled_points_R.tpl
--output-pilot-point-file=modeled_points_R.dat
--real2mod-file=real2mod_points_R.in
--ppk2fac-file=ppk2fac_points_R.in
--fac2real-file=fac2real_points_R.in
--kriging-factors-file=modeled_factors_points_R.bin
--kriged-values-file=modeled_points_R.mod
--modflow-input-file=modeled_R_field.mod
--nan-value-mask=-999
--kriging-structure-file=spec_variogram.str
--variogram-model=culebra
--variogram-search=9500
--max-value=-1
--min-value=-19
--zone-indicator-file=1 zone_recharge.geo
--zone-indicator-value=1 1
--zone-initial-value-eqn=0 0 -30.0
--zone-initial-value-eqn=1 0 -11.0
--no-fixed-in-zone=0
--no-grids-in-zone=0
--no-grids-in-zone=1
--no-wells-in-zone=0
--real2mod-transform=pow10
```

G.7 Run Control Files

G.7.1 Listing – CVS://Tfields::RunControl/RunFieldID.sh

This script checks out the appropriate template file, replaces TFNUM with the field ID, then executes RunReadScript on the resulting file.

```
#!/bin/bash
FIELDID=$1
export PATH=$PATH:$CVSLIB/bin
[ -z "$FIELDID" ] && {
    break;
}

FIELDID=${FIELDID/coord/}
FIELDID=${FIELDID/r/}
FIELDID=r${FIELDID}

HOST=`hostname -s`
HOSTID=

[ $HOST == "alice" ] && { HOSTID="alice"; }
[ $HOST == "ds105" ] && { HOSTID="tethys"; }
[ $HOST == "tethys" ] && { HOSTID="tethys"; }
[ -z "$HOSTID" ] && {
    echo "Error: unrecognized host ID %HOST"
    exit 1
}

# Check out the appropriate template file
cvs -d :ext:alice.sandia.gov:/nfs/data/CVSLIB/Tfields co -p RunControl/tfield_tpl_${HOSTID}.inp >
tfield_tpl_${HOSTID}.inp

# Make the Run directory
mkdir $FIELDID

# Update the Tfield ID name
cat tfield_tpl_${HOSTID}.inp | sed -e "s/TFNUM/$FIELDID/" > $FIELDID/$FIELDID.inp

# Run the calibration
cd $FIELDID
nohup RunReadScript $FIELDID.inp > $FIELDID.nohup &
cd ..
```

G.7.2 Listing – UpdateField.sh

This script checks out the update template file, replaces TFNUM with the field ID, then executes RunReadScript on the resulting file.

```
#!/bin/bash
FIELDID=$1
export PATH=$PATH:$CVSLIB/bin
[ -z "$FIELDID" ] && {
    break;
}

FIELDID=${FIELDID/coord/}
FIELDID=${FIELDID/r/}
FIELDID=r${FIELDID}

HOST=`hostname -s`
HOSTID=

[ $HOST == "alice" ] && { HOSTID="alice"; }
#[ $HOST == "ds105" ] && { HOSTID="tethys"; }
#[ $HOST == "tethys" ] && { HOSTID="tethys"; }
[ -z "$HOSTID" ] && {
    echo "Error: unrecognized host ID %HOST"
    echo " You MUST run this particular run-control script on ALICE"
    exit 1
}

# Check out the appropriate template file
cvs -d :ext:alice.sandia.gov:/nfs/data/CVSLIB/Tfields co -p RunControl/update_tpl_${HOSTID}.inp >
update_tpl_${HOSTID}.inp

# Make the Run directory
mkdir $FIELDID

# Update the Tfield ID name
cat update_tpl_${HOSTID}.inp | sed -e "s/TFNUM/$FIELDID/" > $FIELDID/update_$FIELDID.inp

# Run the calibration
cd $FIELDID
nohup RunReadScript update_$FIELDID.inp > $FIELDID.nohup &
cd ..
```

Appendix H: Run Control Details

The calibration and travel time calculations were executed by the run master (Marissa Reno, 06733) on the ALICE and TETHYS Linux clusters, as described in Section 2 of this document. The following sections of this appendix provide more detailed structure and process listings regarding the run control process.

H.1 CVS Repository Directory Structure

The CVS repository is organized with the goal of segregating the different pieces into larger directories. Run control inputs, scripts, and log files are all contained under the "RunControl" module. All static inputs, including scripts and configuration data, are contained under the "Inputs" module. All auto-generated outputs from the calibration process are contained under the "Outputs" module. The results of the transport calculations and copies of the final fields are stored in the "Results" module. Files relating to this analysis report, or which were used pre-calibration are stored in the "Analysis" module.

Access control was done using standard Unix user/group permissions, but, primarily, CVS' logging features, which track who made what changes to each file, and what those changes were, were used to ensure that proper control was maintained.

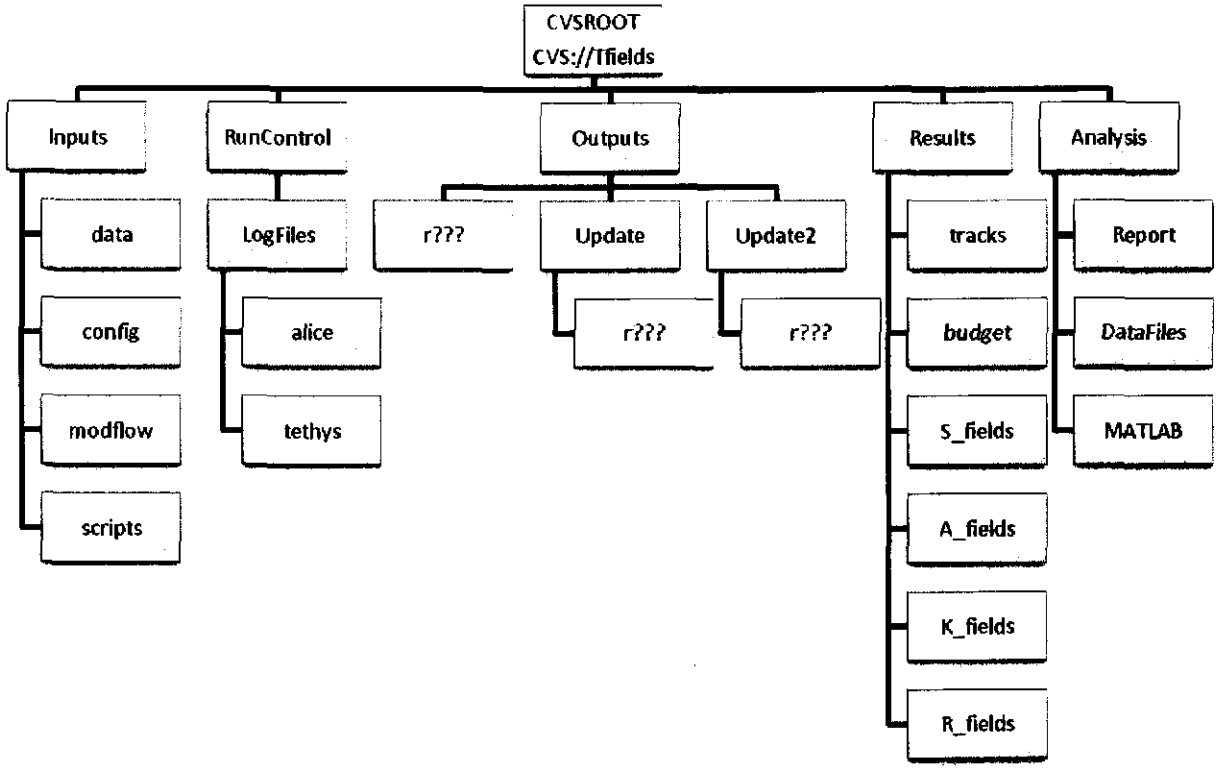


Figure H-1. Layout of the CVS://Tfields repository.

H.2 Executables Used

The following executables and scripts were used during this analysis.

Table H-1. Executables used and scripts qualified under NP 9-1 Appendix C.

Code	Ver	Executable	CVS Module	Qualified
PEST	9.11	pest.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	ppest.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	pslave.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	parrep.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	parcalc.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	svdaprep.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	tempchek.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	fac2real.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	ppk2fac.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
PEST	9.11	mod2obs.exe	CVS://PEST::Builds/Linux	*Under NP 19-1*
MODFLOW2K	1.60	mf2k_1.6.release	CVS://MODFLOW2K::bin/mf2k	*Under NP 19-1*
DTRKMF	1.00	dtrkmf_v0100	CVS://MODFLOW2K::bin/dtrkmf	*Under NP 19-1*
SETUP	*N/A*	setup.sh	CVS://Tfields::Inputs/scripts	*See G.1.1*
INIT2PEST	*N/A*	init2pest.py	CVS://Tfields::Inputs/scripts	*See G.1.2, G.4.1*
MODEL	*N/A*	model.sh	CVS://Tfields::Inputs/scripts	*See G.3.1*
OBS2REAL	*N/A*	obs2real.py	CVS://Tfields::Inputs/scripts	*See G.3.3, G.4.1*
REAL2MOD	*N/A*	real2mod.py	CVS://Tfields::Inputs/scripts	*See G.3.2, G.4.1*
RUNPEST	*N/A*	runpest.sh	CVS://Tfields::Inputs/scripts	*See G.2.1*
PEST_MASTER	*N/A*	template_pmaster.sh	CVS://Tfields::Inputs/scripts	*See G.2.3*
PEST_SLAVE	*N/A*	template_pslave.sh	CVS://Tfields::Inputs/scripts	*See G.2.2*

H.3 Run Control Input and Log Files

The files listed in Table H-2 are present in the "RunControl" module. Please check these files out from CVS if a full listing is desired.

Table H-2. Run control input files, scripts, and log files

File under CVS://Tfields::RunControl	Brief Description
`tfield_tpl_alice.inp`	The input file for calibrations run on ALICE
`tfield_tpl_tethys.inp`	The input file for calibrations run on TETHYS
`update_tpl_alice.inp`	The input file for the 150 fields that required updated heads
`reupdate_tpl_alice.inp`	The input file to run continued calibration after update
`tfield_dtrkmf.inp`	The transport calculation input file for the first 100 fields
`tfield_dtrkmf2.inp`	The transport calculation input file for the remaining fields
`LogFiles/*/*.log`	The raw RunReadScript log file for a calibration run
`LogFiles/alice/r???.rtf`	The RunReadScript output file for a calibration run on alice
`LogFiles/alice/update_r???.rtf`	The RunReadScript output file for an update run on alice
`LogFiles/alice/reupdate_r???.rtf`	The RunReadScript output file for a continued update run

File under CVS://Tfields::RunControl	Brief Description
LogFiles/tethys/r???.rtf	The RunReadScript output file for a calibration run on Tethys
LogFiles/*/r???coordS.o*.?	The *PEST_SLAVE* output file generated by the queue system
LogFiles/*/r???coordM.o*	The *PEST_MASTER* output file generated by the queue sys.
LogFiles/alice/tfield_dtrkmf*.rtf	The travel time calculation log files
RunFieldID.sh	The script to run a calibration
UpdateField.sh	The script to update a calibration
ReUpdateField.sh	The script to continue the update of a calibration
LogFiles/alice/finalize*.*	*These were scripts run initially to correct some of the file check-in errors seen in the r???.rtf log files. However, it was determined thot the file errors were for intermediate files or duplicate files, ond so this was stopped after 29 fields.*

H.4 Calibration Process

The calibration process involved the run master using the "RunFieldID.sh" script to launch a calibration. For the purposes of all examples, field "r207" will be used as an example ID.

- `./RunFieldID.sh r207`

This script checks out the template file based on the current host (alice or tethys) and then replaces the "TFNUM" variable with "r207". The resulting input file is then launched using RunReadScript. The following is an excerpt from the r207.rtf log file, showing the command execution by RunReadScript (items in blue are captured console output, italics are comments by RunReadSCript, remaining are commands issued in the input file):

Excerpt from CVS://Tfields::RunControl/LogFiles/alice/r207.rtf

```
COMMAND mv -f mf2k_1.6.release mf2k.exe
Execute: mv -f mf2k_1.6.release mf2k.exe
COMMAND mv -f dtrkmf_v0100 dtrkmf.exe
Execute: mv -f dtrkmf_v0100 dtrkmf.exe
COMMAND echo $TESTNAM > ReadMe
Execute: echo r207 > ReadMe
COMMAND echo Started >> ReadMe
Execute: echo Started >> ReadMe
COMMAND date >> ReadMe
Execute: date >> ReadMe
COMMAND hostname >> ReadMe
Execute: hostname >> ReadMe
COMMAND ./setup.sh $TESTNAM | tee -a ReadMe
Execute: ./setup.sh r207 | tee -a ReadMe
 Processing observation data
  Reading from "spec_observations.in"
   For test id HEAD
    Added 44 observations in 1 groups
    saved observation instruction file to: modeled_head.ins
   For test id H3
    Added 77 observations in 4 groups
    saved observation instruction file to: modeled_h3.ins
   For test id WIPP13
    Added 185 observations in 9 groups
    saved observation instruction file to: modeled_wipp13.ins
   For test id H11
```

```
    Added 150 observations in 10 groups
    saved observation instruction file to: modeled_h11.ins
   For test id P14
    Added 92 observations in 5 groups
    saved observation instruction file to: modeled_p14.ins
   For test id H19
    Added 157 observations in 7 groups
    saved observation instruction file to: modeled_h19.ins
   For test id WQSP1
    Added 40 observations in 2 groups
    saved observation instruction file to: modeled_wqsp1.ins
   For test id WQSP2
    Added 85 observations in 4 groups
    saved observation instruction file to: modeled_wqsp2.ins
   For test id WIPP11
    Added 274 observations in 12 groups
    saved observation instruction file to: modeled_wipp11.ins
   For test id SNL14
    Added 276 observations in 12 groups
    saved observation instruction file to: modeled_snl14.ins
For the parameter T:
 Reading from "spec_points_T.in"
  5 zones defined for parameter T
   reading r207coord.map == 1 for zone id  1
   reading zone_halitemargins.geo == 1 for zone id  1
   reading zone_dissolution.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_T.inf
  Creating initial values field for parameter T
   Zone 0 assigned 12747 cells with an average value of -4.35012789382
   Zone 1 assigned 27431 cells with an average value of -6.65617460672
   Zone 2 assigned 18539 cells with an average value of -3.32268627924
   Zone 3 assigned 21253 cells with an average value of -11.8607894806
   Zone 4 assigned 7218 cells with an average value of -25.0
   saved initial values field to: modeled_init_points_T.mod
  Creating pilot points for parameter T
   Assigned 225 variable and  18 fixed pilot points to zone 0
   Assigned 255 variable and  32 fixed pilot points to zone 1
   Assigned 91 variable and  11 fixed pilot points to zone 2
   Assigned 0 variable and  0 fixed pilot points to zone 3
   Assigned 0 variable and  0 fixed pilot points to zone 4
   saved pilot points template file to: modeled_points_T.tpl
   saved initial pilot points data to: modeled_points_T.dat
  Creating groundwater-utility program input-redirection files
   saved REAL2MOD template file to: real2mod_points_T.tpl
   saved REAL2MOD input file to: real2mod_points_T.in
   saved FAC2REAL input file to: fac2real_points_T.in
   saved PPK2FAC input file to: ppk2fac_points_T.in
   saved initial pilot points summary to: modeled_init_points_T.tex
For the parameter S:
 Reading from "spec_points_S.in"
  5 zones defined for parameter S
   reading zone_transition.geo == 1 for zone id  1
   reading zone_unconfined.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_S.inf
  Creating initial values field for parameter S
   Zone 0 assigned 39680 cells with an average value of -5.0
   Zone 1 assigned 8031 cells with an average value of -4.0
   Zone 2 assigned 11006 cells with an average value of -1.5
   Zone 3 assigned 21253 cells with an average value of -5.0
   Zone 4 assigned 7218 cells with an average value of -25.0
   saved initial values field to: modeled_init_points_S.mod
  Creating pilot points for parameter S
   Assigned 198 variable and  79 fixed pilot points to zone 0
   Assigned 36 variable and  0 fixed pilot points to zone 1
   Assigned 39 variable and  0 fixed pilot points to zone 2
```

```
    Assigned 0 variable and  0 fixed pilot points to zone 3
    Assigned 0 variable and  0 fixed pilot points to zone 4
    saved pilot points template file to: modeled_points_S.tpl
    saved initial pilot points data to: modeled_points_S.dat
   Creating groundwater-utility program input-redirection files
    saved REAL2MOD template file to: real2mod_points_S.tpl
    saved REAL2MOD input file to: real2mod_points_S.in
    saved FAC2REAL input file to: fac2real_points_S.in
    saved PPK2FAC input file to: ppk2fac_points_S.in
    saved initial pilot points summary to: modeled_init_points_S.tex
 For the parameter A:
  Reading from "spec_points_A.in"
   5 zones defined for parameter A
    reading r207coord.map == 1 for zone id  1
    reading zone_halitemargins.geo == 1 for zone id  1
    reading zone_dissolution.geo == 1 for zone id  2
    reading zone_halite.geo == 1 for zone id  3
    reading zone_noflow.geo == 1 for zone id  4
    saved zone field to: modeled_zones_points_A.inf
   Creating initial values field for parameter A
    Zone 0 assigned 12747 cells with an average value of 0.0
    Zone 1 assigned 27431 cells with an average value of 0.0
    Zone 2 assigned 18539 cells with an average value of 0.0
    Zone 3 assigned 21253 cells with an average value of 0.0
    Zone 4 assigned 7218 cells with an average value of -25.0
    saved initial values field to: modeled_init_points_A.mod
   Creating pilot points for parameter A
    Assigned 135 variable and  0 fixed pilot points to zone 0
    Assigned 152 variable and  0 fixed pilot points to zone 1
    Assigned 30 variable and  0 fixed pilot points to zone 2
    Assigned 0 variable and  0 fixed pilot points to zone 3
    Assigned 0 variable and  0 fixed pilot points to zone 4
    saved pilot points template file to: modeled_points_A.tpl
    saved initial pilot points data to: modeled_points_A.dat
   Creating groundwater-utility program input-redirection files
    saved REAL2MOD template file to: real2mod_points_A.tpl
    saved REAL2MOD input file to: real2mod_points_A.in
    saved FAC2REAL input file to: fac2real_points_A.in
    saved PPK2FAC input file to: ppk2fac_points_A.in
    saved initial pilot points summary to: modeled_init_points_A.tex
 For the parameter R:
  Reading from "spec_points_R.in"
   2 zones defined for parameter R
    reading zone_recharge.geo == 1 for zone id  1
    saved zone field to: modeled_zones_points_R.inf
   Creating initial values field for parameter R
    Zone 0 assigned 86975 cells with an average value of -30.0
    Zone 1 assigned 213 cells with an average value of -11.0
    saved initial values field to: modeled_init_points_R.mod
   Creating pilot points for parameter R
    Assigned 0 variable and  0 fixed pilot points to zone 0
    Assigned 3 variable and  1 fixed pilot points to zone 1
    saved pilot points template file to: modeled_points_R.tpl
    saved initial pilot points data to: modeled_points_R.dat
   Creating groundwater-utility program input-redirection files
    saved REAL2MOD template file to: real2mod_points_R.tpl
    saved REAL2MOD input file to: real2mod_points_R.in
    saved FAC2REAL input file to: fac2real_points_R.in
    saved PPK2FAC input file to: ppk2fac_points_R.in
    saved initial pilot points summary to: modeled_init_points_R.tex
  saved PEST control file to: r207_init.pst
  saved PEST control file to: r207_once.pst
  saved PEST run management file: r207_init.rmf
  saved PEST run management file: r207_svda.rmf
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave1
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
```

```
Copied files to slave2
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave3
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave4
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave5
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave6
COMMAND ./runpest.sh $TESTNAM 6
Execute: ./runpest.sh r207 6
Starting in 3 . 2 . 1 . done!
Your job 1247.2-6:1 ("r207coordS") has been submitted.
Your job 1248 ("r207coordM") has been submitted.
Job 1248 exited with exit code 0.
The job r207coordS of user(s) mdrenc does not exist
............
COMMAND echo Finished >> ReadMe
Execute: echo Finished >> ReadMe
COMMAND date >> ReadMe
Execute: date >> ReadMe
```

The last section of blue text, it shows that the job "r207coordS" was submitted as *job 1247.2-6*. The files CVS://Tfields::RunControl/LogFiles/alice/r207coordS.o1247.* are the console logs from the PEST_SLAVE jobs. The file CVS://Tfields::RunControl/LogFiles/alice/r207coordM.o1248 is the PEST_MASTER log file. Each run-control log file will contain the job number of the queued jobs which allows the user to find the full log of the PEST commands. The line "Job 1248 exited with exit code 0." indicates that the calibration was able to run and complete successfully. A non-zero exit code indicates an error.

H.5 Update Calibration Process

The update calibration process was done in the same manner as the original calibration, but a different script was used ("UpdateField.sh") in order to select the *update_tpl_alice.inp* input file. The following is a selection from the "update_r207.rtf" log file.

```
COMMAND mv -f mf2k_1.6.release mf2k.exe
Execute: mv -f mf2k_1.6.release mf2k.exe
COMMAND mv -f dtrkmf_v0100 dtrkmf.exe
Execute: mv -f dtrkmf_v0100 dtrkmf.exe
COMMAND mv modeled_points_T.dat modeled_final_points_T.dat
Execute: mv modeled_points_T.dat modeled_final_points_T.dat
COMMAND mv modeled_points_A.dat modeled_final_points_A.dat
Execute: mv modeled_points_A.dat modeled_final_points_A.dat
COMMAND mv modeled_points_R.dat modeled_final_points_R.dat
Execute: mv modeled_points_R.dat modeled_final_points_R.dat
COMMAND mv modeled_points_S.dat modeled_final_points_S.dat
Execute: mv modeled_points_S.dat modeled_final_points_S.dat
COMMAND echo $TESTNAM > ReadMe
Execute: echo r207 > ReadMe
COMMAND echo Started >> ReadMe
Execute: echo Started >> ReadMe
COMMAND date >> ReadMe
Execute: date >> ReadMe
COMMAND hostname >> ReadMe
Execute: hostname >> ReadMe
COMMAND ./setup.sh $TESTNAM | tee -a ReadMe
Execute: ./setup.sh r207 | tee -a ReadMe
 Processing observation data
  Reading from "spec_observations.in"
   For test id HEAD
    Added 44 observations in 1 groups
```

```
    saved observation instruction file to: modeled_head.ins
   For test id H3
    Added 77 observations in 4 groups
    saved observation instruction file to: modeled_h3.ins
   For test id WIPP13
    Added 185 observations in 9 groups
    saved observation instruction file to: modeled_wipp13.ins
   For test id H11
    Added 150 observations in 10 groups
    saved observation instruction file to: modeled_h11.ins
   For test id P14
    Added 92 observations in 5 groups
    saved observation instruction file to: modeled_p14.ins
   For test id H19
    Added 157 observations in 7 groups
    saved observation instruction file to: modeled_h19.ins
   For test id WQSP1
    Added 40 observations in 2 groups
    saved observation instruction file to: modeled_wqsp1.ins
   For test id WQSP2
    Added 85 observations in 4 groups
    saved observation instruction file to: modeled_wqsp2.ins
   For test id WIPP11
    Added 274 observations in 12 groups
    saved observation instruction file to: modeled_wipp11.ins
   For test id SNL14
    Added 276 observations in 12 groups
    saved observation instruction file to: modeled_snl14.ins
For the parameter T:
 Reading from "spec_points_T.in"
  5 zones defined for parameter T
   reading r207coord.map == 1 for zone id  1
   reading zone_halitemargins.geo == 1 for zone id  1
   reading zone_dissolution.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_T.inf
  Creating initial values field for parameter T
   Zone 0 assigned 12747 cells with an average value of -4.35012789382
   Zone 1 assigned 27431 cells with an average value of -6.65617460672
   Zone 2 assigned 18539 cells with an average value of -3.32268627924
   Zone 3 assigned 21253 cells with an average value of -11.8607894806
   Zone 4 assigned 7218 cells with an average value of -25.0
   saved initial values field to: modeled_init_points_T.mod
  Creating pilot points for parameter T
   Assigned 225 variable and  18 fixed pilot points to zone 0
   Assigned 255 variable and  32 fixed pilot points to zone 1
   Assigned 91 variable and  11 fixed pilot points to zone 2
   Assigned 0 variable and  0 fixed pilot points to zone 3
   Assigned 0 variable and  0 fixed pilot points to zone 4
   saved pilot points template file to: modeled_points_T.tpl
   saved initial pilot points data to: modeled_points_T.dat
  Creating groundwater-utility program input-redirection files
   saved REAL2MOD template file to: real2mod_points_T.tpl
   saved REAL2MOD input file to: real2mod_points_T.in
   saved FAC2REAL input file to: fac2real_points_T.in
   saved PPK2FAC input file to: ppk2fac_points_T.in
   saved initial pilot points summary to: modeled_init_points_T.tex
For the parameter S:
 Reading from "spec_points_S.in"
  5 zones defined for parameter S
   reading zone_transition.geo == 1 for zone id  1
   reading zone_unconfined.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_S.inf
  Creating initial values field for parameter S
   Zone 0 assigned 39680 cells with an average value of -5.0
   Zone 1 assigned 8031 cells with an average value of -4.0
```

```
    Zone 2 assigned 11006 cells with an average value of -1.5
    Zone 3 assigned 21253 cells with an average value of -5.0
    Zone 4 assigned 7218 cells with an average value of -25.0
    saved initial values field to: modeled_init_points_S.mod
   Creating pilot points for parameter S
    Assigned 198 variable and  79 fixed pilot points to zone 0
    Assigned 36 variable and  0 fixed pilot points to zone 1
    Assigned 39 variable and  0 fixed pilot points to zone 2
    Assigned 0 variable and  0 fixed pilot points to zone 3
    Assigned 0 variable and  0 fixed pilot points to zone 4
    saved pilot points template file to: modeled_points_S.tpl
    saved initial pilot points data to: modeled_points_S.dat
   Creating groundwater-utility program input-redirection files
    saved REAL2MOD template file to: real2mod_points_S.tpl
    saved REAL2MOD input file to: real2mod_points_S.in
    saved FAC2REAL input file to: fac2real_points_S.in
    saved PPK2FAC input file to: ppk2fac_points_S.in
    saved initial pilot points summary to: modeled_init_points_S.tex
For the parameter A:
 Reading from "spec_points_A.in"
  5 zones defined for parameter A
   reading r207coord.map == 1 for zone id  1
   reading zone_halitemargins.geo == 1 for zone id  1
   reading zone_dissolution.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_A.inf
  Creating initial values field for parameter A
   Zone 0 assigned 12747 cells with an average value of 0.0
   Zone 1 assigned 27431 cells with an average value of 0.0
   Zone 2 assigned 18539 cells with an average value of 0.0
   Zone 3 assigned 21253 cells with an average value of 0.0
   Zone 4 assigned 7218 cells with an average value of -25.0
   saved initial values field to: modeled_init_points_A.mod
  Creating pilot points for parameter A
   Assigned 135 variable and  0 fixed pilot points to zone 0
   Assigned 152 variable and  0 fixed pilot points to zone 1
   Assigned 30 variable and  0 fixed pilot points to zone 2
   Assigned 0 variable and  0 fixed pilot points to zone 3
   Assigned 0 variable and  0 fixed pilot points to zone 4
   saved pilot points template file to: modeled_points_A.tpl
   saved initial pilot points data to: modeled_points_A.dat
  Creating groundwater-utility program input-redirection files
   saved REAL2MOD template file to: real2mod_points_A.tpl
   saved REAL2MOD input file to: real2mod_points_A.in
   saved FAC2REAL input file to: fac2real_points_A.in
   saved PPK2FAC input file to: ppk2fac_points_A.in
   saved initial pilot points summary to: modeled_init_points_A.tex
For the parameter R:
 Reading from "spec_points_R.in"
  2 zones defined for parameter R
   reading zone_recharge.geo == 1 for zone id  1
   saved zone field to: modeled_zones_points_R.inf
  Creating initial values field for parameter R
   Zone 0 assigned 86975 cells with an average value of -30.0
   Zone 1 assigned 213 cells with an average value of -11.0
   saved initial values field to: modeled_init_points_R.mod
  Creating pilot points for parameter R
   Assigned 0 variable and  0 fixed pilot points to zone 0
   Assigned 3 variable and  1 fixed pilot points to zone 1
   saved pilot points template file to: modeled_points_R.tpl
   saved initial pilot points data to: modeled_points_R.dat
  Creating groundwater-utility program input-redirection files
   saved REAL2MOD template file to: real2mod_points_R.tpl
   saved REAL2MOD input file to: real2mod_points_R.in
   saved FAC2REAL input file to: fac2real_points_R.in
   saved PPK2FAC input file to: ppk2fac_points_R.in
   saved initial pilot points summary to: modeled_init_points_R.tex
 saved PEST control file to: r207_init.pst
```

```
  saved PEST control file to: r207_once.pst
  saved PEST run management file: r207_init.rmf
  saved PEST run management file: r207_svda.rmf
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave1
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave2
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave3
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave4
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave5
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave6
COMMAND ./fix_snl8.sh $TESTNAM | tee -a ReadMe
Execute: ./fix_snl8.sh r207 | tee -a ReadMe
 PARREP Version 9.11. Watermark Numerical Computing.

 Reading parameter value file r207_last.par ----->
 Data for 1309 parameters read from file r207_last.par.

 Reading file r207_init.pst and writing file r207_update.pst ----->
 File r207_update.pst written ok.
 PARAMFIX Version 9.11. Watermark Numerical Computing.

 Reading parameter fix file paramfix_points.in ....
 - data for 1120 parameters read from file paramfix_points.in.

 Reading file r207_update.pst and writing file r207_init.pst ....
 - file r207_update.pst read ok.
 - file r207_init.pst written ok.
mv: cannot stat `r207_new.pst': No such file or directory
COMMAND ./runpest.sh $TESTNAM 6
Execute: ./runpest.sh r207 6
Starting in 3 . 2 . 1 . done!
Your job 2567.2-6:1 ("r207coordS") has been submitted.
Your job 2568 ("r207coordM") has been submitted.
Job 2568 exited with exit code 0.
The job r207coordS of user(s) mdrenc does not exist
............
```

The reader can see the insertion of the "fix_snl8" command, which limits the update calibration to the area shown in Figure 4-3 and discussed in Section 4.2. In this case, the *PEST_SLAVE* and *PEST_MASTER* output logs are in CVS://Tfields::RunControl/LogFiles/alice/r207coordS.o2567.? and CVS://Tfields::RunControl/LogFiles/alice/r207coordM.o2568 respectively.

H.6 Re-Update Calibration Process

A few files were selected for continued recalibration, as discussed in Section 4.2.2 of this document. These fields were launched by the run master using the "ReUpdateField.sh" script, which operates in the same way as "RunFieldID.sh" and "UpdateField.sh". Again, an excerpt from the log file "reupdate_r207.rtf" is presented here.

```
COMMAND mv -f mf2k_1.6.release mf2k.exe
Execute: mv -f mf2k_1.6.release mf2k.exe
```

```
COMMAND mv -f dtrkmf_v0100 dtrkmf.exe
Execute: mv -f dtrkmf_v0100 dtrkmf.exe
COMMAND mv modeled_points_T.dat modeled_final_points_T.dat
Execute: mv modeled_points_T.dat modeled_final_points_T.dat
COMMAND mv modeled_points_A.dat modeled_final_points_A.dat
Execute: mv modeled_points_A.dat modeled_final_points_A.dat
COMMAND mv modeled_points_R.dat modeled_final_points_R.dat
Execute: mv modeled_points_R.dat modeled_final_points_R.dat
COMMAND mv modeled_points_S.dat modeled_final_points_S.dat
Execute: mv modeled_points_S.dat modeled_final_points_S.dat
COMMAND echo $TESTNAM > ReadMe
Execute: echo r207 > ReadMe
COMMAND echo Started >> ReadMe
Execute: echo Started >> ReadMe
COMMAND date >> ReadMe
Execute: date >> ReadMe
COMMAND hostname >> ReadMe
Execute: hostname >> ReadMe
COMMAND ./setup10.sh $TESTNAM | tee -a ReadMe
Execute: ./setup10.sh r207 | tee -a ReadMe
 Processing observation data
  Reading from "spec_observations.in"
   For test id HEAD
    Added 44 observations in 1 groups
    saved observation instruction file to: modeled_head.ins
   For test id H3
    Added 77 observations in 4 groups
    saved observation instruction file to: modeled_h3.ins
   For test id WIPP13
    Added 185 observations in 9 groups
    saved observation instruction file to: modeled_wipp13.ins
   For test id H11
    Added 150 observations in 10 groups
    saved observation instruction file to: modeled_h11.ins
   For test id P14
    Added 92 observations in 5 groups
    saved observation instruction file to: modeled_p14.ins
   For test id H19
    Added 157 observations in 7 groups
    saved observation instruction file to: modeled_h19.ins
   For test id WQSP1
    Added 40 observations in 2 groups
    saved observation instruction file to: modeled_wqsp1.ins
   For test id WQSP2
    Added 85 observations in 4 groups
    saved observation instruction file to: modeled_wqsp2.ins
   For test id WIPP11
    Added 274 observations in 12 groups
    saved observation instruction file to: modeled_wipp11.ins
   For test id SNL14
    Added 276 observations in 12 groups
    saved observation instruction file to: modeled_snl14.ins
 For the parameter T:
  Reading from "spec_points_T.in"
   5 zones defined for parameter T
    reading r207coord.map == 1 for zone id  1
    reading zone_halitemargins.geo == 1 for zone id  1
    reading zone_dissolution.geo == 1 for zone id  2
    reading zone_halite.geo == 1 for zone id  3
    reading zone_noflow.geo == 1 for zone id  4
    saved zone field to: modeled_zones_points_T.inf
   Creating initial values field for parameter T
    Zone 0 assigned 12747 cells with an average value of -4.35012789382
    Zone 1 assigned 27431 cells with an average value of -6.65617460672
    Zone 2 assigned 18539 cells with an average value of -3.32268627924
    Zone 3 assigned 21253 cells with an average value of -11.8607894806
    Zone 4 assigned 7218 cells with an average value of -25.0
    saved initial values field to: modeled_init_points_T.mod
   Creating pilot points for parameter T
    Assigned 225 variable and  18 fixed pilot points to zone 0
    Assigned 255 variable and  32 fixed pilot points to zone 1
```

```
    Assigned 91 variable and  11 fixed pilot points to zone 2
    Assigned 0 variable and  0 fixed pilot points to zone 3
    Assigned 0 variable and  0 fixed pilot points to zone 4
    saved pilot points template file to: modeled_points_T.tpl
    saved initial pilot points data to: modeled_points_T.dat
   Creating groundwater-utility program input-redirection files
    saved REAL2MOD template file to: real2mod_points_T.tpl
    saved REAL2MOD input file to: real2mod_points_T.in
    saved FAC2REAL input file to: fac2real_points_T.in
    saved PPK2FAC input file to: ppk2fac_points_T.in
    saved initial pilot points summary to: modeled_init_points_T.tex
For the parameter S:
 Reading from "spec_points_S.in"
  5 zones defined for parameter S
   reading zone_transition.geo == 1 for zone id  1
   reading zone_unconfined.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_S.inf
  Creating initial values field for parameter S
   Zone 0 assigned 39680 cells with an average value of -5.0
   Zone 1 assigned 8031 cells with an average value of -4.0
   Zone 2 assigned 11006 cells with an average value of -1.5
   Zone 3 assigned 21253 cells with an average value of -5.0
   Zone 4 assigned 7218 cells with an average value of -25.0
   saved initial values field to: modeled_init_points_S.mod
  Creating pilot points for parameter S
   Assigned 198 variable and  79 fixed pilot points to zone 0
   Assigned 36 variable and  0 fixed pilot points to zone 1
   Assigned 39 variable and  0 fixed pilot points to zone 2
   Assigned 0 variable and  0 fixed pilot points to zone 3
   Assigned 0 variable and  0 fixed pilot points to zone 4
   saved pilot points template file to: modeled_points_S.tpl
   saved initial pilot points data to: modeled_points_S.dat
  Creating groundwater-utility program input-redirection files
   saved REAL2MOD template file to: real2mod_points_S.tpl
   saved REAL2MOD input file to: real2mod_points_S.in
   saved FAC2REAL input file to: fac2real_points_S.in
   saved PPK2FAC input file to: ppk2fac_points_S.in
   saved initial pilot points summary to: modeled_init_points_S.tex
For the parameter A:
 Reading from "spec_points_A.in"
  5 zones defined for parameter A
   reading r207coord.map == 1 for zone id  1
   reading zone_halitemargins.geo == 1 for zone id  1
   reading zone_dissolution.geo == 1 for zone id  2
   reading zone_halite.geo == 1 for zone id  3
   reading zone_noflow.geo == 1 for zone id  4
   saved zone field to: modeled_zones_points_A.inf
  Creating initial values field for parameter A
   Zone 0 assigned 12747 cells with an average value of 0.0
   Zone 1 assigned 27431 cells with an average value of 0.0
   Zone 2 assigned 18539 cells with an average value of 0.0
   Zone 3 assigned 21253 cells with an average value of 0.0
   Zone 4 assigned 7218 cells with an average value of -25.0
   saved initial values field to: modeled_init_points_A.mod
  Creating pilot points for parameter A
   Assigned 135 variable and  0 fixed pilot points to zone 0
   Assigned 152 variable and  0 fixed pilot points to zone 1
   Assigned 30 variable and  0 fixed pilot points to zone 2
   Assigned 0 variable and  0 fixed pilot points to zone 3
   Assigned 0 variable and  0 fixed pilot points to zone 4
   saved pilot points template file to: modeled_points_A.tpl
   saved initial pilot points data to: modeled_points_A.dat
  Creating groundwater-utility program input-redirection files
   saved REAL2MOD template file to: real2mod_points_A.tpl
   saved REAL2MOD input file to: real2mod_points_A.in
   saved FAC2REAL input file to: fac2real_points_A.in
   saved PPK2FAC input file to: ppk2fac_points_A.in
```

```
    saved initial pilot points summary to: modeled_init_points_A.tex
 For the parameter R:
  Reading from "spec_points_R.in"
   2 zones defined for parameter R
    reading zone_recharge.geo == 1 for zone id  1
    saved zone field to: modeled_zones_points_R.inf
   Creating initial values field for parameter R
    Zone 0 assigned 86975 cells with an average value of -30.0
    Zone 1 assigned 213 cells with an average value of -11.0
    saved initial values field to: modeled_init_points_R.mod
   Creating pilot points for parameter R
    Assigned 0 variable and  0 fixed pilot points to zone 0
    Assigned 3 variable and  1 fixed pilot points to zone 1
    saved pilot points template file to: modeled_points_R.tpl
    saved initial pilot points data to: modeled_points_R.dat
   Creating groundwater-utility program input-redirection files
    saved REAL2MOD template file to: real2mod_points_R.tpl
    saved REAL2MOD input file to: real2mod_points_R.in
    saved FAC2REAL input file to: fac2real_points_R.in
    saved PPK2FAC input file to: ppk2fac_points_R.in
    saved initial pilot points summary to: modeled_init_points_R.tex
  saved PEST control file to: r207_init.pst
  saved PEST control file to: r207_once.pst
  saved PEST run management file: r207_init.rmf
  saved PEST run management file: r207_svda.rmf
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave1
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave2
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave3
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave4
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave5
cp: cannot stat `dtrkmf_manypts.inp': No such file or directory
cp: cannot stat `dtrkmf_tracks.inp': No such file or directory
Copied files to slave6
COMMAND ./update_pest.sh $TESTNAM | tee -a ReadMe
Execute: ./update_pest.sh r207 | tee -a ReadMe
 PARREP Version 9.11. Watermark Numerical Computing.

 Reading parameter value file r207_last.par ----->
 Data for 1309 parameters read from file r207_last.par.

 Reading file r207_init.pst and writing file r207_update.pst ----->
 File r207_update.pst written ok.
COMMAND ./runpest.sh $TESTNAM 6
Execute: ./runpest.sh r207 6
Starting in 3 . 2 . 1 . done!
Your job 2760.2-6:1 ("r207coordS") has been submitted.
Your job 2761 ("r207coordM") has been submitted.
Job 2761 exited with exit code 0.
The job r207coordS of user(s) mdreno does not exist
............
```

Instead of "fix_snl8.sh", the "update_pest.sh" script was used to correct the input file for continued recalibration.

H.7 Travel Time Calculation Process

The travel times were calculated on the final calibrated fields using "tfields_dtrkmf.inp" and "tfields_dtrkmf2.inp". The only difference between these files is the list of fields on which they operate, in order to run transport calculations on all the fields.

Unlike the calibration runs, these were run directly using RunReadScript by the run master, and not done using a template input file and scripts. The commands issued were:

- `nohup RunReadScript tfield_dtrkmf.inp`
- `nohup RunReadScript tfield_dtrimf2.inp`

Both these runs were executed on the ALICE system, as DTRKMF was qualified on ALICE. An excerpt from the log file is presented below.

```
COMMAND date
Execute: date
Tue Jun 30 13:56:07 MDT 2009
COMMAND chmod +x run_dtrkmf.sh
Execute: chmod +x run_dtrkmf.sh
COMMAND ln dtrkmf_v0100 dtrkmf.exe
Execute: ln dtrkmf_v0100 dtrkmf.exe
COMMAND ./run_dtrkmf.sh SelectedFields.txt
Execute: ./run_dtrkmf.sh SelectedFields.txt
================================================================
Checking out Outputs/Update/r440/modeled_K_field.mod
RCS:  /nfs/data/CVSLIB/Tfields/Outputs/Update/r440/modeled_K_field.mod,v
VERS: 1.1
***************
================================================================
Checking out Outputs/Update/r440/modeled_A_field.mod
RCS:  /nfs/data/CVSLIB/Tfields/Outputs/Update/r440/modeled_A_field.mod,v
VERS: 1.1
***************
================================================================
Checking out Outputs/Update/r440/modeled_S_field.mod
RCS:  /nfs/data/CVSLIB/Tfields/Outputs/Update/r440/modeled_S_field.mod,v
VERS: 1.1
***************
================================================================
Checking out Outputs/Update/r440/modeled_R_field.mod
RCS:  /nfs/data/CVSLIB/Tfields/Outputs/Update/r440/modeled_R_field.mod,v
VERS: 1.1
***************
================================================================
Checking out Outputs/Update/r440/modeled_flow.bud
RCS:  /nfs/data/CVSLIB/Tfields/Outputs/Update/r440/modeled_flow.bud,v
VERS: 1.1
***************
Normal DTRKMF Completion
  ######USING TREADWAY ANALYTIC SOLVER######
  PARTICLE EXIT SUMMARY:    1  5.0732E+03  3.9565E+03  1.3063E+04  1.8439E+04
Normal DTRKMF Completion
  ######USING TREADWAY ANALYTIC SOLVER######
  PARTICLE EXIT SUMMARY:    1  2.1351E+04  1.8060E+04  1.1209E+04  3.0590E+04
================================================================
.....
.....
.....
```

The "run_dtrkmf.sh" command is found in CVS://Tfields::Inputs/scripts, and is presented in G.5.2.