555318

Analysis Report for Preparation of 2010 Culebra Potentiometric Surface Contour Map

Revision 2

Task Number: 1.4.2.3

Report Date: 4/6/2011

Author:	[illegible] Kristopher L. Kuhlman, 6212 Repository Performance Department	4/6/2011 Date
Technical Review:	[illegible] Kevin S. Barnhart, 6212 Repository Performance Department	4/6/2011 Date
QA Review:	[illegible] Mario J. Chavez, 6210 Carlsbad Programs Group	4/6/11 Date
Management Review:	[illegible] Christi Leigh, 6212 Manager, Repository Performance Department	4/6/2011 Date

WIPP:1.4.2.3:TD:QA-L:RECERT:549085

Table of Contents

1 Introduction

This report documents the preparation of a potentiometric contour map and particle track for the Culebra Member of the Rustler Formation in the vicinity of the Waste Isolation Pilot Plant (WIPP), for inclusion in the 2011 Annual Site Environmental Report (ASER), which summarizes 2010 data. The driver for this analysis is the draft of the Stipulated Final Order sent to the New Mexico Environment Department (NMED) on May 28, 2009 (Moody, 2009). This Analysis Report follows the procedure laid out in SP 09-09 (Kuhlman, 2009), which is based upon this NMED driver. This report is a revision of Kuhlman (2010), where the same analysis is being performed on February 2010 data, rather than June, 2009 data.

Beginning with the ensemble of 100 calibrated MODFLOW transmissivity (T), horizontal anisotropy (A), and areal recharge (R) fields (Hart et al., 2009) used in WIPP performance assessment (PA), three average parameter fields are used as input to MODFLOW to simulate freshwater heads within and around the WIPP land withdrawal boundary (LWB). PEST is used to adjust a subset of the boundary conditions in the ensemble-average model to obtain the best-fit match between the observed freshwater heads from February 2010 and the model-predicted heads. The output of the averaged, PEST-calibrated MODFLOW model is both contoured and used to compute an advective particle track forward from the WIPP waste handling shaft.

2 Scientific Approach

2.1 Overview

Steady-state groundwater flow simulations are carried out using much the same software and approach used in the analysis report for AP-114 Task 7 (Hart et al., 2009) to create the calibrated fields used as inputs – see Table 1 for a summary of all software used in this analysis. The MODFLOW parameter fields (including transmissivity (T), anisotropy (A), and recharge (R)) used here are an ensemble average of the Culebra parameter fields used for WIPP PA in the CRA-2009 performance assessment baseline calculations (PABC). To clearly distinguish between the two MODFLOW models, the original MODFLOW model, which consists of 100 realizations of calibrated parameter fields (Hart et al., 2009), will be referred to as the "PA MODFLOW model." The model derived here from the PA MODFLOW model, used to construct the resulting contour map and particle track, is referred to as the "averaged MODFLOW model." The calibrated model T, A and R input fields, model boundary conditions, and other model input files are appropriately averaged across all 100 calibrated realizations to produce a single averaged steady-state MODFLOW flow model that can be used to predict regional Culebra groundwater flow across the WIPP site.

The calibration process that resulted in the 100 model realizations of the PA MODFLOW model used PEST to adjust spatially variable model parameters, while assuming fixed MODFLOW boundary conditions. The calibration targets for the PA MODFLOW model were both snapshots of undisturbed heads across the site and transient head responses to large-scale pumping tests. Hart et al. (2009) describe the forward model setup and PEST calibration effort for the CRA-2009 PABC. An analogous but much simpler process is used in the averaged MODFLOW model; here PEST is used to modify a subset of the MODFLOW boundary conditions (see boundaries marked in red on Figure 1). The calibration targets for PEST associated with the average MODFLOW model are the February 2010 freshwater heads at Culebra monitoring wells. Boundary conditions are modified while holding spatially variable model parameters (T, A, and R) constant; in the calibration of the PA MODFLOW model, the boundary conditions were fixed, while adjusting the spatially variable parameters.

Table 1. Software used

Software	Version	Description	
ODFLOW-2000	1.6	Flow model	Acquired; qualified under NP 19-1 (Harbaugh et al., 2000)
PEST	9.11	Inverse model	Developed; qualified under NP 19-1 (Doherty, 2002)
DTRKMF	1.00	Particle tracker	Developed; qualified under NP 19-1
Python	2.3.4	Scripting Language (plotting)	Commercial off the shelf

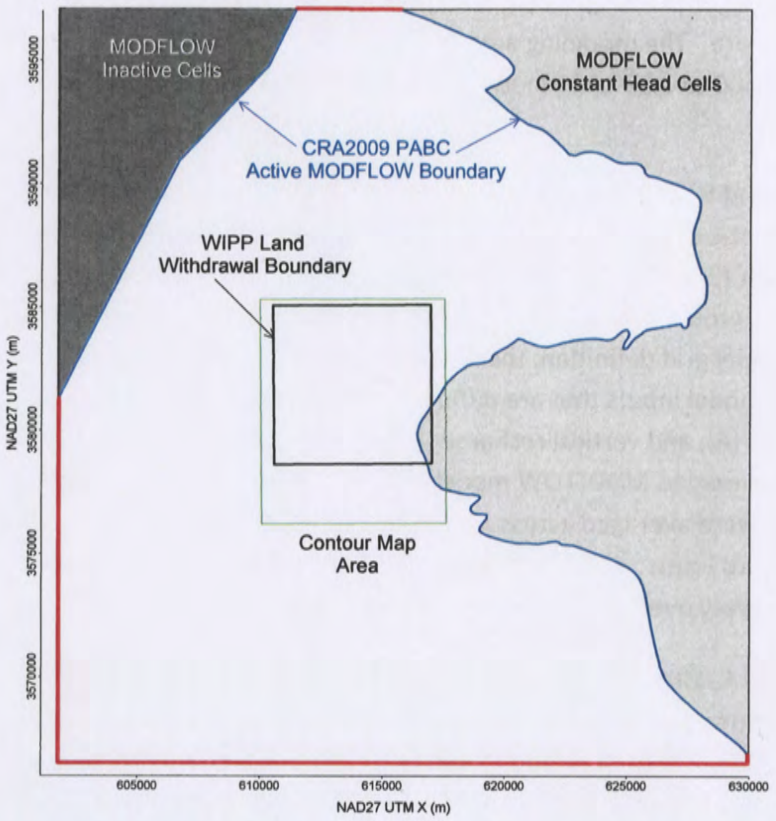

Figure 1. MODFLOW-2000 model domain, adjusted boundary conditions shown in red, contour area outlined in green.

The resulting heads from the PEST-calibrated ensemble-average flow model are contoured over an area surrounding the WIPP site using the matplotlib Python graphics library (a subset of the complete MODFLOW model domain – see the green rectangle surrounding the WIPP LWB in Figure 1). The track made by a conservative (i.e., non-dispersive and non-reactive) particle released from the waste handling shaft to the WIPP land withdrawal boundary is computed from the resulting flow field in MODFLOW using DTRKMF, and also plotted with matplotlib. Scatter plot statistics summarizing the fit of the PEST-calibrated model to the observed freshwater head at Culebra monitoring wells are created in matplotlib. MODFLOW, PEST, DTRKMF, and the Bash and Python scripts written for this work were executed on the

PA Linux cluster (`alice.sandia.gov`), while the plotting and creation of figures was done using Python matplotlib scripts on an Intel-Xeon-equipped desktop computer running Ubuntu Linux 9.04.

2.2 Creating Ensemble Average MODFLOW Simulation

An ensemble-average MODFLOW model is used to compute both the freshwater head and flow vectors across the model domain; the heads are then contoured and the cell-by-cell flow vectors are used to compute particle tracks. The ensemble-averaged inputs are used to create a single average simulation that produces a single output, rather than averaging the 100 individual outputs of the Culebra flow model used for WIPP PA.

The MODFLOW model grid is a single layer, comprised of 307 rows and 284 columns, each model cell being 100 meters square. The modeling area spans 601,700 to 630,000 meters in the east-west direction, and 3,566,500 to 3,597,100 meters in the north-south direction, both in UTM NAD27 coordinates (zone 13).

The calibrated T, A, and R parameter fields from the PA MODFLOW model were checked out of the CVS repository using the `checkout_average_run_modflow.sh` script (all scripts are listed completely in the Appendix; input files are available on the attached media). The model inputs can be divided into two groups. The first group is the model inputs that are the same across all 100 calibrated realizations; these include the model grid definition, the boundary conditions, and the model solver parameters. The second group is the model inputs that are different for each realization; these include transmissivity (T), horizontal anisotropy (A), and vertical recharge (R). The constant model inputs in the first group are used directly in the averaged MODFLOW model (checked out from the CVS repository), while the inputs in the second group were averaged across all 100 calibrated model realizations using the Python script `average_realizations.py`. All three averaged parameters were log transformed before being averaged, since they vary over multiple orders of magnitude.

2.3 Boundary Conditions

The boundary conditions taken from the PA MODFLOW model are used as the initial condition from which PEST calibration proceeds. There are two types of boundary conditions in both MODFLOW models. The first type of condition includes geologic or hydrologic boundaries, which correspond to known physical features in the flow domain. The no-flow boundary along the axis of Nash Draw is a hydrologic boundary (i.e., the boundary along the dark gray region in Figure 1). Also, the constant-head boundary along the halite margin corresponds to a geologic boundary (i.e., the eastern irregular boundary adjoining the light gray region in Figure 1).

Physical boundaries are believed to be well known, and are not adjusted in the PEST calibration. The second type of boundary condition includes the constant-head cells along the rest of the model domain; the linear southern, southwestern, and northern boundaries that coincide with the rectangular frame surrounding the model domain are all of this type (shown as a heavy red line in Figure 1). The value of specified head used along this second boundary type is adjusted in the PEST calibration process.

The Python script `boundary_types.py` is used to distinguish between the two different types of specified head boundary conditions based on the specified head value used in the PA MODFLOW model. All constant-head cells (specified by a value of -1 in the MODFLOW IBOUND array from the PA MODFLOW model) that have a starting head value greater than 1000 m (corresponding to the land surface) are left fixed and not adjusted in the PEST optimization. The remaining constant-head cells are distinguished by setting their IBOUND array value to -2 (which is still interpreted as a constant-head value by MODFLOW, but allows simpler discrimination between boundary conditions in scripts elsewhere).

Using the output from `boundary_types.py`, the Python script `surface_02_extrapolate.py` computes the heads at active (IBOUND=1) and adjustable constant-head boundary condition cells (IBOUND=-2), given parameter values for the surface to extrapolate.

2.4 PEST Calibration of Averaged MODFLOW Model to Observations

There are three major types of inputs to PEST. The first type of input includes the set of February 2010 freshwater head values used as targets for the PEST calibration. The second class of inputs includes the entire MODFLOW model setup derived from the PA MODFLOW model and described in the previous section, along with any pre- or post-processing scripts or programs needed; this comprises the forward model that PEST runs repeatedly to estimate sensitivities of model outputs to model inputs. The third type of input includes the PEST configuration files, which include parameter and observation groups, indicating which parameters in the MODFLOW model will be adjusted in the inverse simulation.

Freshwater head values used as targets for the PEST calibration were collected in February 2010 (Waterson, 2011) and are summarized in Table 2.

Table 2. Calibration targets used in PEST, from Watterson (2011).

Well	Measurement Date	Freshwater head elevation (ft	Freshwater head elevation (m AMSL)	Water Density
AEC-7	02/09/10	3065.10	934.242	1.080
C-2737 (PIP)	02/10/10	3022.51	921.262	1.027
ERDA-9	02/10/10	3033.60	924.642	1.070
H-02b2	02/10/10	3045.97	928.411	1.011
H-03b2	02/10/10	3014.90	918.943	1.042
H-04bR	02/08/10	3008.06	916.857	1.018
H-05b	02/09/10	3082.24	939.466	1.096
H-06bR	02/08/10	3071.78	936.278	1.037
H-07b1	02/08/10	2998.55	913.959	1.006
H-09c (PIP)	02/08/10	2998.59	913.971	1.006
H-10c	02/09/10	3028.11	922.969	1.091
H-11b4	02/09/10	3006.87	916.493	1.060
H-12	02/09/10	3008.00	916.838	1.097
H-15R	02/10/10	3018.86	920.149	1.120
H-16	02/10/10	3047.09	928.754	1.039
H-17	02/08/10	3008.66	917.039	1.135
H-19b0	02/10/10	3015.14	919.016	1.067
IMC-461	02/08/10	3045.85	928.376	1.007
SNL-01	02/09/10	3083.66	939.899	1.030
SNL-02	02/08/10	3072.56	936.516	1.008
SNL-03	02/09/10	3082.66	939.594	1.032
SNL-05	02/08/10	3075.58	937.438	1.009
SNL-08	02/09/10	3052.63	930.443	1.093
SNL-09	02/08/10	3055.05	931.179	1.018
SNL-10	02/09/10	3054.46	930.999	1.009
SNL-12	02/08/10	3003.62	915.502	1.004
SNL-13	02/09/10	3011.86	918.014	1.025
SNL-14	02/08/10	3005.84	916.179	1.046
SNL-16	02/08/10	3010.49	917.597	1.015
SNL-17	02/09/10	3006.36	916.337	1.005
SNL-18	02/09/10	3075.17	937.311	1.005
SNL-19	02/08/10	3072.98	936.644	1.007
WIPP-11	02/10/10	3082.54	939.557	1.037
WIPP-13	02/10/10	3077.56	938.039	1.045
WIPP-19	02/10/10	3064.23	933.978	1.051
WQSP-1	02/10/10	3075.76	937.491	1.046
WQSP-2	02/10/10	3083.53	939.860	1.045
WQSP-3	02/10/10	3073.40	936.774	1.144
WQSP-4	02/10/10	3015.72	919.190	1.074
WQSP-5	02/10/10	3013.44	918.496	1.025
WQSP-6	02/10/10	3025.16	922.068	1.014

To minimize the number of estimable parameters, and to ensure a degree of smoothness in the constant-head boundary condition values, a parametric surface is used to extrapolate the heads to the estimable boundary conditions. The surface is of the same form described in the analysis report for AP-114 Task 7. The parametric surface is given by the following equation:

$$h(x,y) = A + B\,(y + D\,\mathrm{sign}(y)\,\mathrm{abs}(y)^{\alpha}) + C\,(E\,x^3 + F\,x^2 - x)$$

where sign(y) is the function returning 1 for $y>0$, -1 for $y<0$ and 0 for $y=0$ and x and y are coordinates scaled to the range $-1 \le \{x,y\} \le 1$. In Hart et al. (2009), the values A=928.0, B=8.0, C=1.2, D=1.0, α=0.5, E=1.0, and F=-1.0 are used with the above equation.

PEST was then used to estimate the values of parameters *A, B, C, D, E, F,* and α given the observed heads in Table 2. The Python script `surface_02_extrapolate.py` was used to compute the MODFLOW starting head input file (which is also used to specify the constant-head values) from the parameters *A-F* and *exponent*. Each forward run of the forward model corresponded to a call to the Bash script `run_02_model`. This script called the `surface_02_extrapolate.py` script, the MODFLOW-2000 v1.6 executable, and the qualified PEST utility `mod2obs.exe`, which is used to extract and interpolate model-predicted heads from the MODFLOW output files at observation well locations.

The PEST-specific input files (the third type of input) were generated from the observed heads using the Python script `create_pest_02_input.py`. The PEST input files include the instruction file (how to read the model output), the template files (how to write the model input files), and the PEST control file (listing the ranges and initial values for the estimable parameters and the weights associated with observations).

2.5 Figures Generated from Calibrated MODFLOW Model

The MODFLOW model is run predictively using the ensemble-averaged model parameters, along with the PEST-calibrated boundary conditions. The resulting cell-by-cell flow budget is then used by DTRKMF to compute a particle track from the waste-handling shaft to at least the edge of the WIPP land withdrawal boundary. The Python script `convert_dtrkmf_output_for_surfer.py` converts the IJK cell-based results of DTRKMF into a UTM x and y coordinate system, saving the results in the Surfer blanking file format to facilitate plotting with Surfer. The heads in the binary MODFLOW output file are converted to an ASCII Surfer grid format using the Python script `head_bin2ascii.py`.

The resulting particle track and contours of the model-predicted head are plotted using Surfer 9 for an area including the WIPP land withdrawal boundary, similar to the region shown in previous versions of the ASER (e.g., see Figure 6.11 in DOE (2008)), see green outline in Figure 1. The modeled heads extracted from the MODFLOW output by `mod2obs.exe` are then merged into a common file for plotting using the Python script `merge_observed_modeled_heads.py`.

3 Results

3.1 Freshwater Head Contours

The model-generated freshwater head contours in Figure 2 and Figure 3 show the known characteristics of groundwater flow in the Culebra at the WIPP site. There is a roughly east-west trending band of steeper gradients, corresponding to known lower transmissivities. The uncontoured region in the eastern part of the figure corresponds to the portion of the Culebra that is located stratigraphically between halite in other members of the Rustler Formation (Tamarisk Member above and Los Medaños Member below). This region east of the "halite margin" is represented as having high head but extremely low permeability, essentially serving as a no-flow boundary in this area.

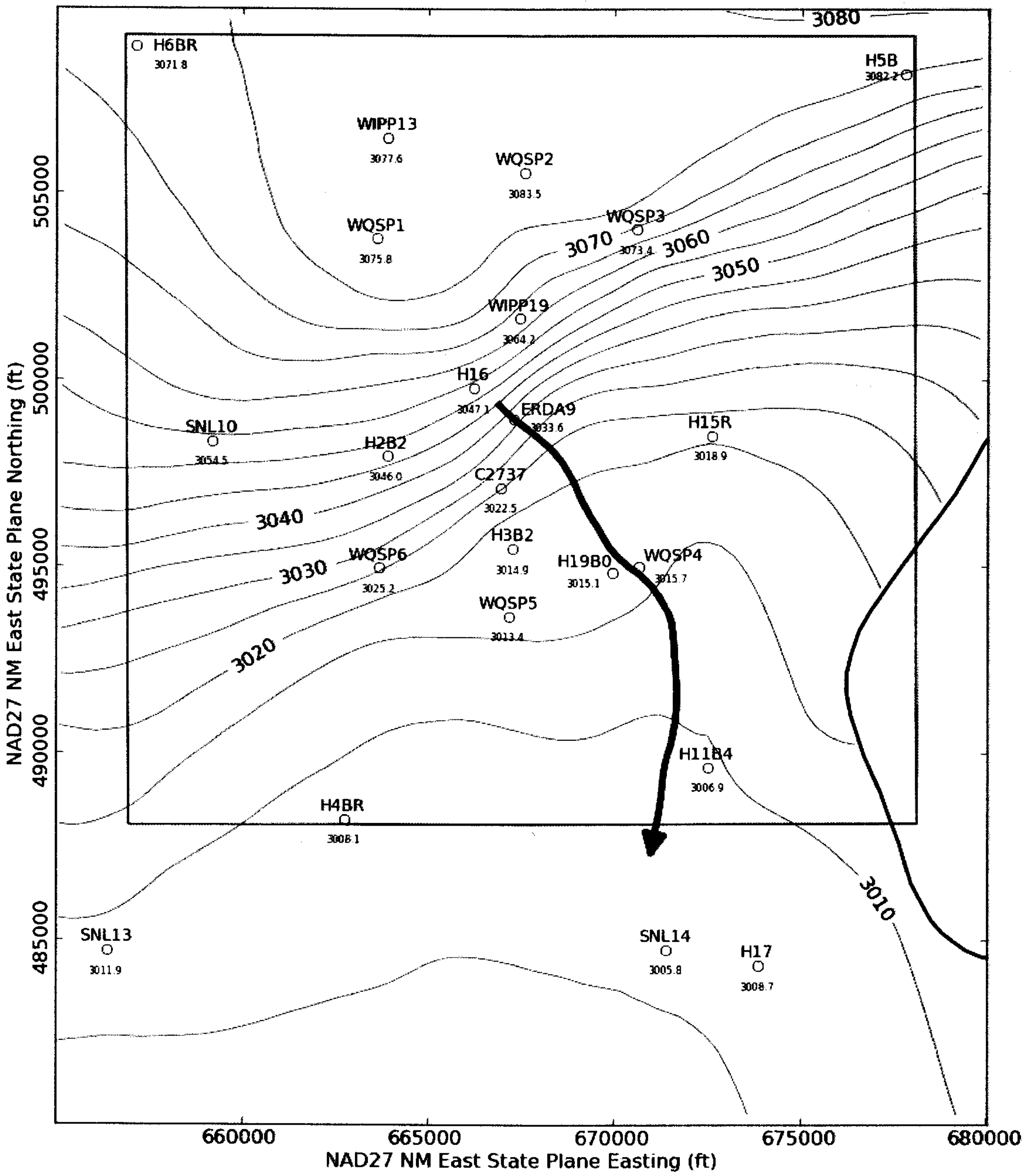

Figure 2. Model-generated February 2010 freshwater head contours with observed head listed at each well (5-foot contour interval) with blue water particle track from waste handling shaft to WIPP LWB

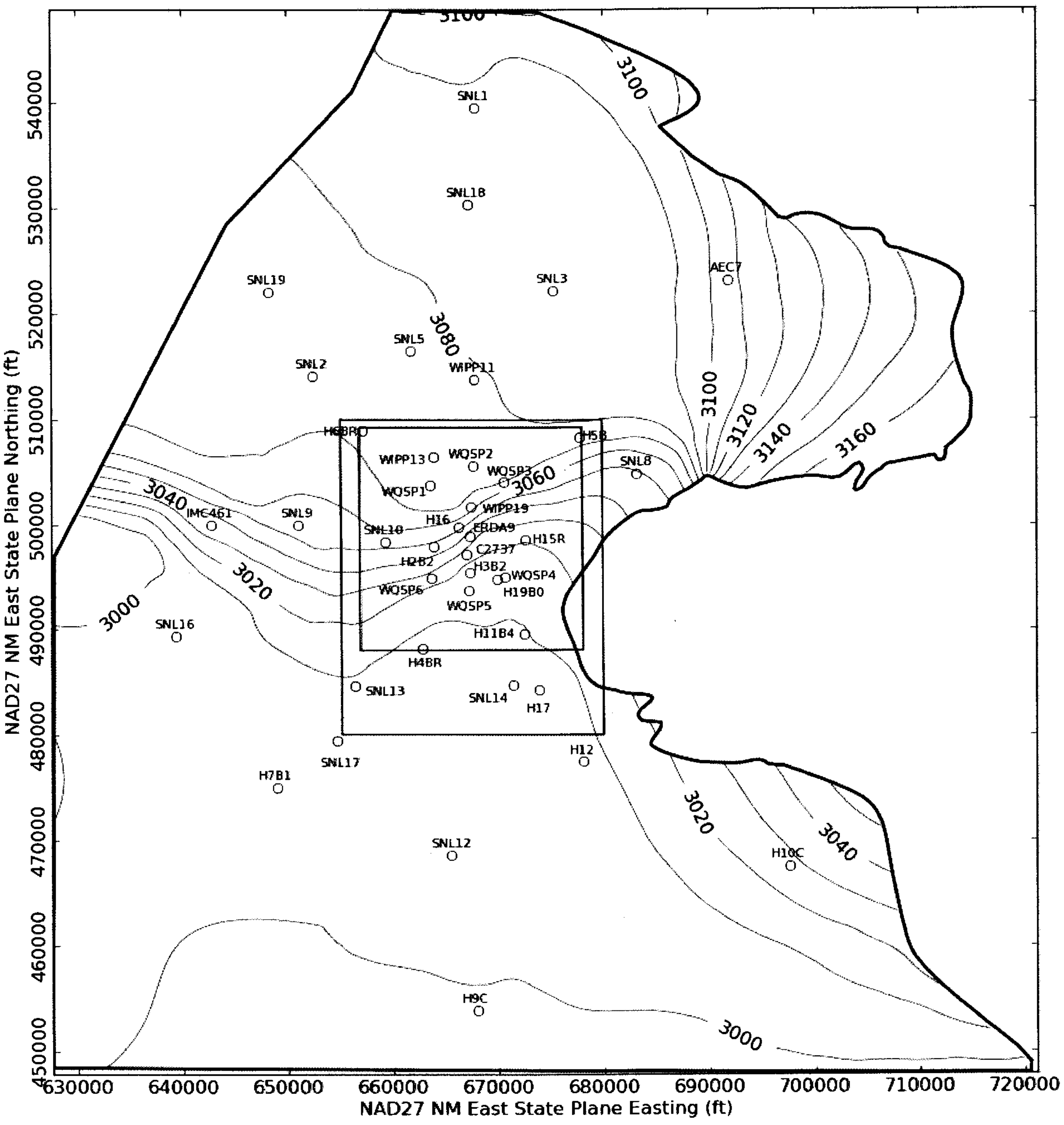

Figure 3. MODFLOW-modeled heads for entire model domain (10-foot contour interval). Green rectangle indicates region contoured in Figure 2, black square is WIPP LWB.

3.2 Particle Track

The heavy blue line in Figure 2 shows the DTRKMF-predicted path a water particle would take through the Culebra from the coordinates corresponding to the WIPP waste handling shaft to the land withdrawal boundary (a DTRKMF-computed path length of 4.075 km). Assuming a thickness of 4 m for the transmissive portion of the Culebra and a constant porosity of 16%, the travel time to the WIPP LWB is 6283 years (output from DTRKMF is adjusted from an original 7.75-m Culebra thickness), for an average velocity of 0.65 m/yr.

3.3 Measured vs. Modeled Fit

The scatter plot in Figure 4 shows measured and modeled freshwater heads at the observation locations used in the PEST calibration. The observations are divided into three groups, based on proximity to the WIPP site. Wells within the LWB are represented by red crosses, wells outside but within 3 km of the LWB are represented with green "×"s, and other wells within the MODFLOW model domain but distant from the WIPP site are given by a blue asterisk. These groupings were utilized in the PEST calibration; higher weights (2.5) were given to wells inside the LWB, and lower weights (0.4) were given to wells distant to the WIPP site, while wells in the middle received an intermediate weight (1.0); AEC-7 was given a low weight (0.01), to prevent its large residual from dominating the optimization. Weights were determined through trial and error. Additional observations representing the average heads north of the LWB and south of the LWB were used to help prevent over-smoothing of the estimated results across the LWB. This allowed PEST to improve the fit of the model to observed heads inside the area contoured in Figure 2, at the expense of fitting wells closer to the boundary conditions (i.e., wells not shown in Figure 2).

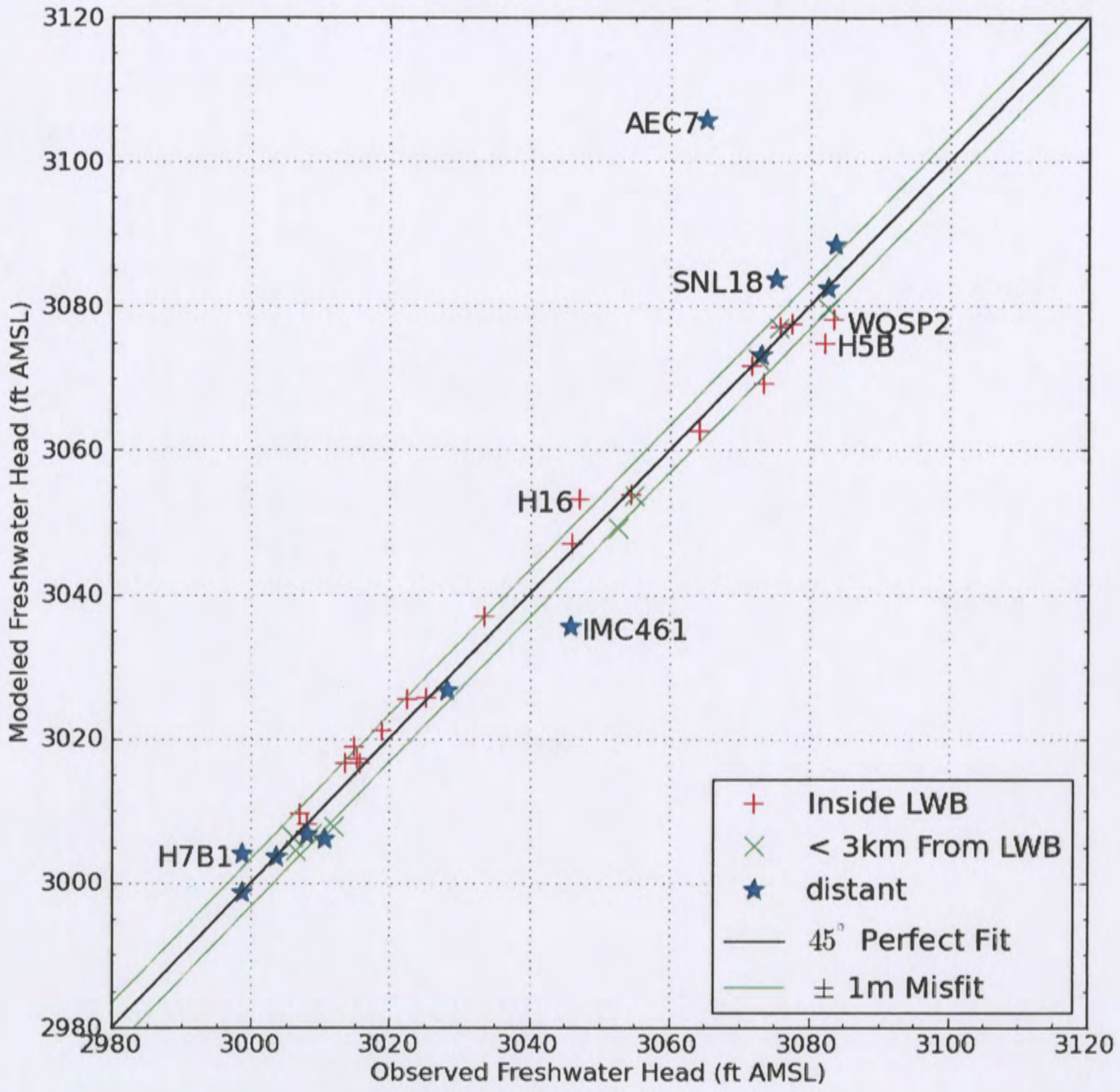

Figure 4. Measured vs. modeled scatter plot for PEST-calibrated MODFLOW-2000 generated heads and February 2010 observed freshwater heads

The central diagonal line in Figure 4 represents a perfect model fit (1:1 or 45-degree slope); the two lines on either side of this represent a 1-m misfit above or below the perfect fit. Wells more than 1.5 m from the 1:1 line are labeled. AEC-7 has a large misfit (12.4 m), for two reasons. First, this well has historically had an anomalously low freshwater head elevation, lower than wells around it in all directions. Secondly, it did not have a May 2007 observation (due to ongoing well reconfiguration activities) and therefore was not included as a calibration target in the PA MODFLOW model calibration. The ensemble-average T, A, and R fields used here were not calibrated to accommodate this observation. This well is situated in a low-transmissivity region, and near the constant-head boundary associated with the halite margin, therefore PEST will not be able to improve this fit solely through adjustment of the boundary conditions indicated with red in Figure 1.

The squared correlation coefficient (R^2) for the measured vs. modeled data is listed in Table 3. Figure 5 and Figure 6 show the distribution of errors resulting from the PEST-adjusted fit to observed data. The wells within and near the WIPP LWB have an R^2 of greater than 99%. The distribution in Figure 5 is roughly symmetric about 0, indicating there is not a strong bias.

Table 3. Measured vs. Modeled correlation coefficients

dataset	measured vs. modeled squared correlation coefficient
wells inside WIPP LWB	0.9911
wells <3km from WIPP LWB	0.9903
all wells	0.9467

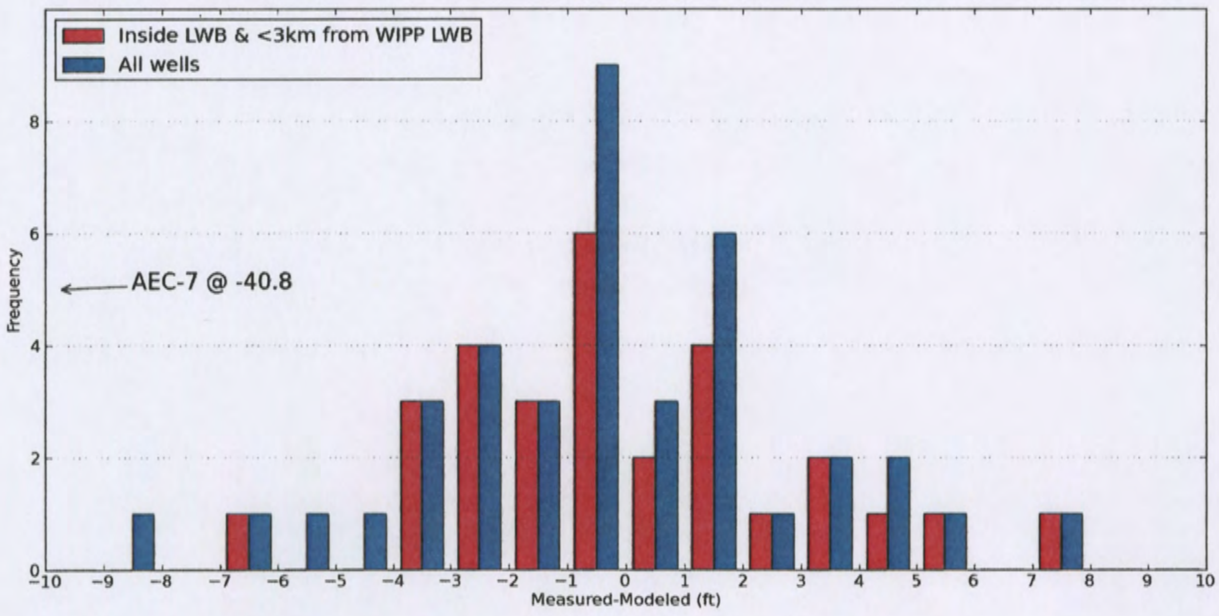

Figure 5. Histogram of Measured-Modeled errors

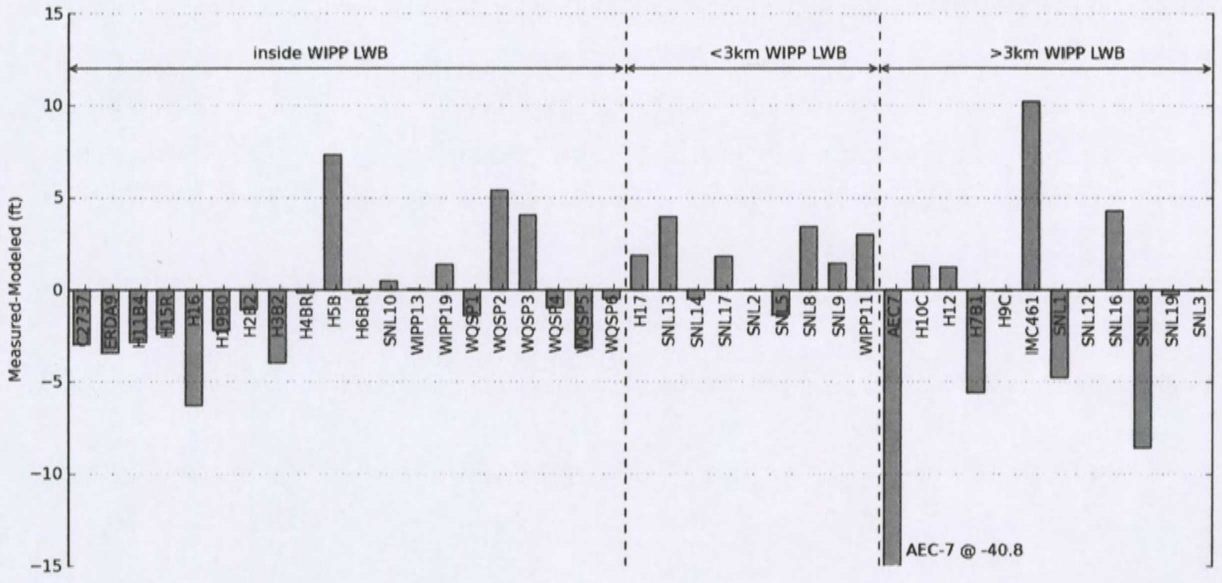

Figure 6. Measured-Modeled errors for each well

Aside from AEC-7, and to a lesser degree some other distant wells whose modeled values do not greatly impact the contours shown in Figure 2, the model fit to the February 2010 observations is very good. The ensemble-average model captures the average Culebra behavior, while the PEST calibration improved the model fit to the specific February 2010 observations.

4 References

DOE (Department of Energy). 2008. *Waste Isolation Pilot Plant Annual Site Environmental Report for 2007*. DOE/WIPP-08-2225.

Doherty, J. 2002. *PEST: Model Independent Parameter Estimation*. Watermark Numerical Computing, Brisbane, Australia.

Harbaugh, A.W., E.R. Banta, M.C. Hill, and M.G. McDonald. 2000. *MODFLOW-2000, the U.S. Geological Survey modular ground-water model – User guide to modularization concepts and the Ground-Water Flow Process*. U.S. Geological Survey Open-File Report 00-92.

Hart, D.B., S.A. McKenna, and R.L. Beauheim. 2009. *Analysis Report for Task 7 of AP-114: Calibration of Culebra Transmissivity Fields*. Carlsbad, NM, Sandia National Laboratories, ERMS 552391.

Kuhlman, K.L. 2010. Analysis Report for Preparation of 2009 Culebra Potentiometric Surface Contour Map, Rev 1, Sandia National Laboratories, Carlsbad, NM, ERMS 552005.

Kuhlman, K.L. 2009. Procedure SP 9-9, revision 0, Preparation of Culebra potentiometric surface contour maps. Carlsbad, NM, Sandia National Laboratories, ERMS 552306.

Moody, D.C. 2009. *Stipulated Final Order for Notice of Violation for Detection Monitoring Program*, Sandia National Laboratories, Carlsbad, NM. WIPP Records Center, ERMS 551713.

Watterson, D. 2011. February 2010 Culebra ASER map data, Washington TRU Solutions, Carlsbad, NM. WIPP Records Center, ERMS 555127.

5 Run Control Narrative

This section is a narrative describing the calculation process mentioned in the text, which produced the figures given there.

Figure 7 gives an overview of the driver script `checkout_average_run_modflow.sh` (§A-4.1); this script first exports the 4 parameter fields (transmissivity (T), anisotropy (A), recharge (R), and storativity (S)) from CVS for each of the 100 realizations of MODFLOW, listed in the file `keepers` (see lines 17-26 of script). Some of the realizations are inside the `Update` or `Update2` subdirectories in CVS, which complicates the directory structure. An equivalent list `keepers_short` is made from `keepers`, and the directories are moved to match the flat directory structure (lines 31-53). At this point, the directory structure has been modified but the MODFLOW input files checked out from CVS are unchanged.

Python script `average_realizations.py` (§A-4.2) is called, which first reads in the `keepers_short` list, then reads in each of the 400 input files and computes the arithmetic average of the base-10 logarithm of the value at each cell across the 100 realizations. The 400 input files are saved as a flattened 2D matrix, in row-major order. The exponentiated result is saved in 4 parameter fields, each with the extension `.avg` instead of `.mod`. A single value from each file, corresponding to either the cell in the southeast corner of the domain (input file row 87188 = model row 307, model column 284 for K and A) or on the west edge of the domain (input file row 45157 = model row 161, model column 1 for R and S) is saved in the text file `parameter_representative_values.txt` to allow checking the calculation in Excel, comparing the results to the value given at the same row of the `.avg` file. The value in the right column of Table 4 can be found by taking the geometric average of the values in the text file, which are the values from the indicated line of each of the 100 realizations.

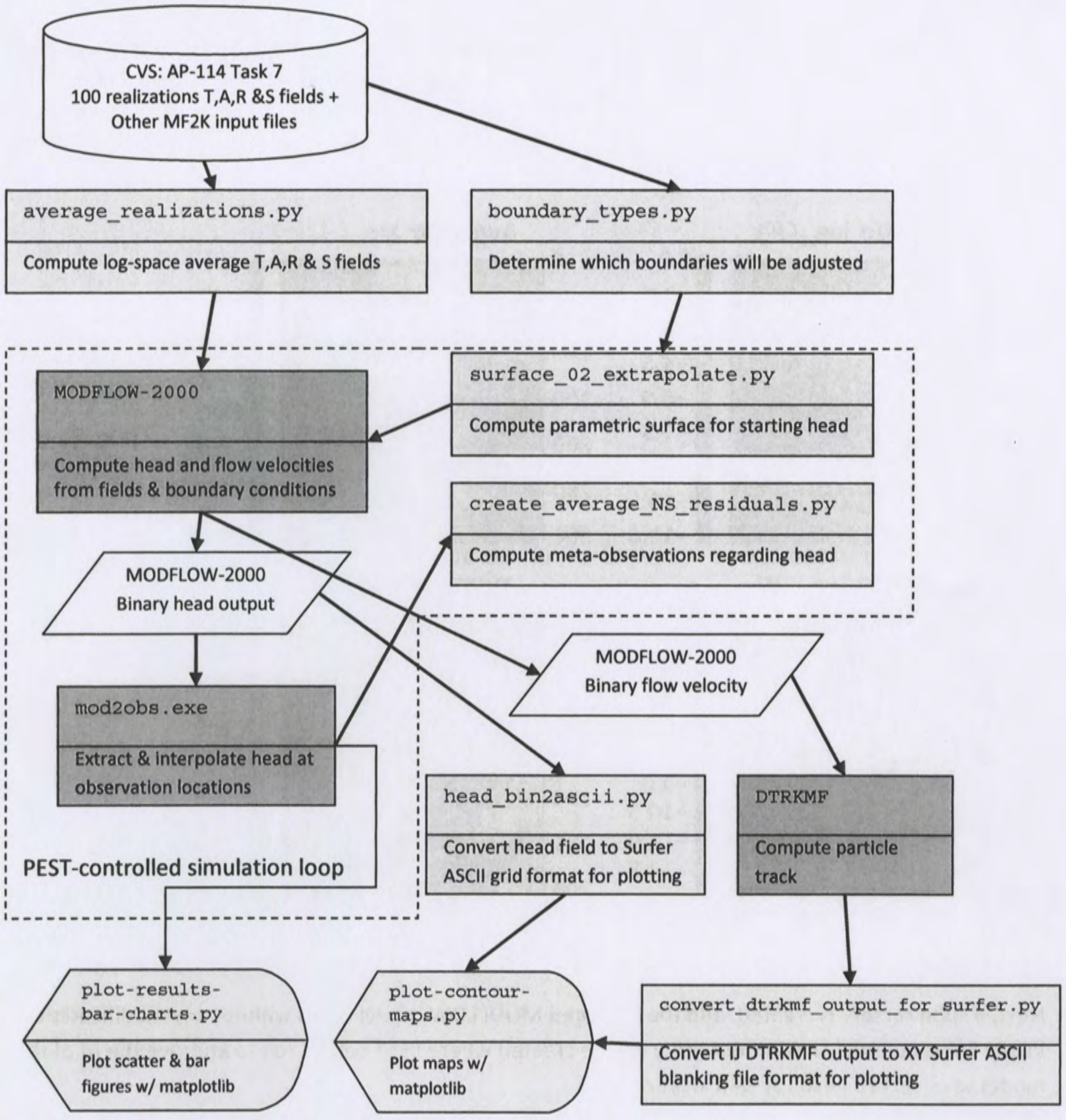
CVS: AP-114 Task 7
100 realizations T,A,R &S fields +
Other MF2K input files
average_realizations.py
Compute log-space average T,A,R & S fields
boundary_types.py
Determine which boundaries will be adjusted
MODFLOW-2000
Compute head and flow velocities from fields & boundary conditions
surface_02_extrapolate.py
Compute parametric surface for starting head
create_average_NS_residuals.py
Compute meta-observations regarding head
MODFLOW-2000
Binary head output
MODFLOW-2000
Binary flow velocity
mod2obs.exe
Extract & interpolate head at observation locations
PEST-controlled simulation loop
head_bin2ascii.py
Convert head field to Surfer ASCII grid format for plotting
DTRKMF
Compute particle track
plot-results-bar-charts.py
Plot scatter & bar figures w/ matplotlib
plot-contour-maps.py
Plot maps w/ matplotlib
convert_dtrkmf_output_for_surfer.py
Convert IJ DTRKMF output to XY Surfer ASCII blanking file format for plotting

Figure 7. Process flowchart; dark gray indicates qualified programs, light gray are scripts written for this analysis

Table 4. Averaged values for representative model cells

Field	Input file row	Model row	Model column	Geometric average
K	87188	307	284	9.2583577E-09
A	87188	307	284	9.6317478E-01
R	45157	161	1	1.4970689E-19
S	45157	161	1	4.0388352E-03

Figure 8 shows plots of the average log10 parameters, which compare with similar figures in Hart et al. (2009); inactive regions <1.0E-15 were reset to 1.0 to improve the plotted color scale. The rest of the calculations are done with these averaged fields.

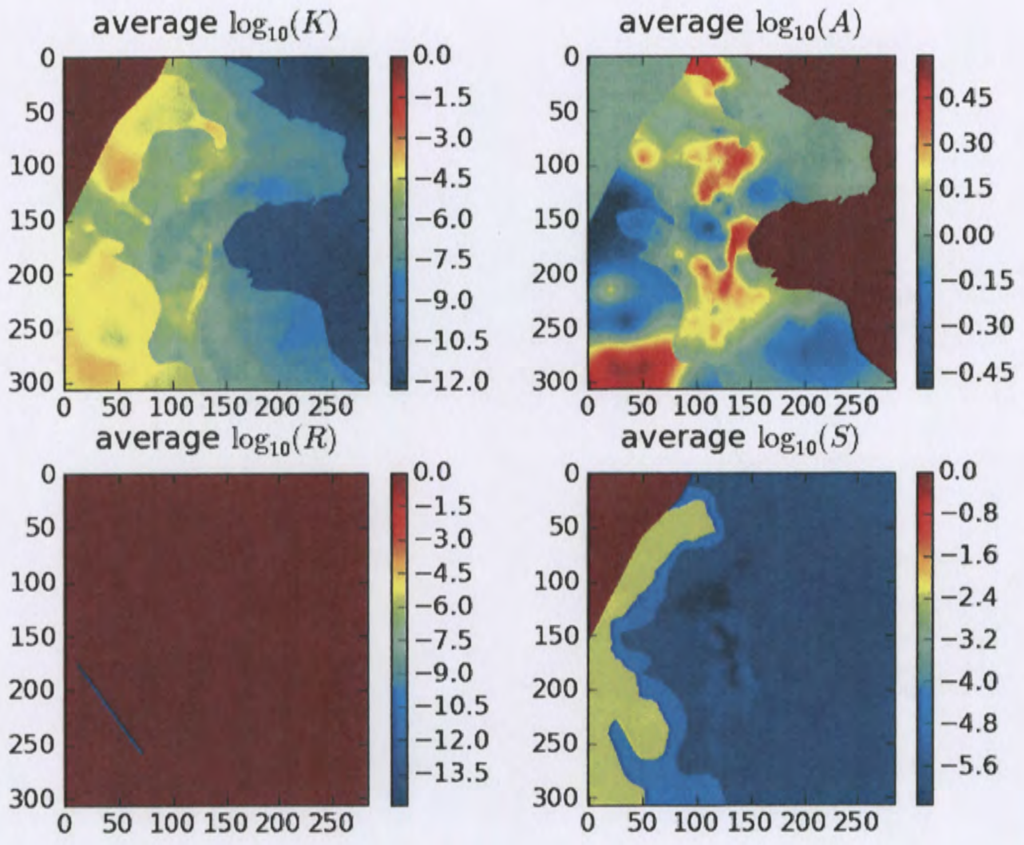

Figure 8. Plots of base-10 logarithms of average parameter fields; rows and columns are labeled on edges of figures.

Next, a subdirectory is created, and the averaged MODFLOW model is run without any modifications by PEST. Subsequently, another directory will be created where PEST will be run to improve the fit of the model to observed heads at well locations.

The next portion of the driving script `checkout_average_run_modflow.sh` links copies of the input files needed to run MODFLOW-2000 and DTRKMF into the `original_average` run directory. Then MODFLOW-2000 is run with the name file `mf2k_head.nam`, producing binary head (`modeled_head.bin`) and binary cell-by-cell flow budget (`modeled_flow.bud`) files, as well as a text listing file (`modeled_head.lst`). DTRKMF is then run with the input files `dtrkmf.in` and `wippctrl.inp`, which utilizes the cell-by-cell budget file written by MODFLOW to generate a particle track output file, `dtrk.out`. The input file `wippctrl.inp` specifies the starting location of the particle in DTRKMF face-centered cell coordinates, the porosity of the aquifer (here 16%), and the coordinates of the corners of the WIPP LWB, since the calculation stops when the particle reaches the LWB.

The Python script `head_bin2ascii.py` (§A-4.7) converts the MODFLOW binary head file, which includes the steady-state head at every element in the flow model domain (307 rows × 284 columns) into a Surfer ASCII grid file format. This file is simply contoured in Surfer, no interpolation or gridding is needed. The Python script `convert_dtrkmf_output_for_surfer.py` (§A-4.9) reads the DTRKMF output file `dtrk.out` and does two things. First it converts the row, column format of this output file to an X,Y format suitable for plotting, and second it converts the effective thickness of the Culebra from 7.75m to 4m. The following table shows the first 10 lines of the `dtrk.out` and the corresponding output of the Python script `dtrk_output_original_average.bln`. The first three columns of `dtrk.out` (top half of Table 5) after the header are cumulative time (red), column (blue), and row (green). The three columns in the blanking file (second half of Table 5) after the header are UTM NAD27 X (blue), UTM NAD27 Y (green), and adjusted cumulative time (red, which is faster faster than the original cumulative travel time by the factor 7.75/4=1.9375). The conversion from row, column to X, Y is

$$X = 601700.0 + 100.0 * column$$
$$Y = 3597100.0 - 100.0 * row \quad ,$$

since the I,J origin is the northwest corner of the model domain (601700,3597100), while the X,Y origin is the southwest corner of the domain. The blanking file is plotted directly in Surfer, since it now has the same coordinates as the ASCII head file.

Table 5. Comparison of first 10 lines of DTRKMF output and converted Surfer blanking file for `original_average`

```
               1          159
  0.00000000E+00  118.79 150.21 1.18790000E+04 1.50210000E+04 0.00000000E+00 1.85168267E-01 1.59999996E-01 1.00000000E+00
  5.53946616E+01  118.86 150.29 1.18859872E+04 1.50285080E+04 1.02562574E+01 1.85130032E-01 1.59999996E-01 1.00000000E+00
  1.10789323E+02  118.93 150.36 1.18929942E+04 1.50359947E+04 2.05104788E+01 1.85094756E-01 1.59999996E-01 1.00000000E+00
  1.66017959E+02  119.00 150.43 1.19000000E+04 1.50434379E+04 3.07321029E+01 1.85062532E-01 1.59999996E-01 1.00000000E+00
  3.27990509E+02  119.21 150.62 1.19206651E+04 1.50624751E+04 5.88294962E+01 1.73534671E-01 1.59999996E-01 1.00000000E+00
  4.89963060E+02  119.42 150.81 1.19415109E+04 1.50813473E+04 8.69490492E+01 1.73684593E-01 1.59999996E-01 1.00000000E+00
  6.51450155E+02  119.62 151.00 1.19624759E+04 1.51000000E+04 1.15010608E+02 1.73860152E-01 1.59999996E-01 1.00000000E+00
  7.40581455E+02  119.75 151.10 1.19749757E+04 1.51102419E+04 1.31170520E+02 1.81333000E-01 1.59999996E-01 1.00000000E+00
  8.29712755E+02  119.87 151.20 1.19874963E+04 1.51204665E+04 1.47335525E+02 1.81390626E-01 1.59999996E-01 1.00000000E+00
```
```
159,1
613579.0,3582079.0,0.00000000e+00
613586.0,3582071.0,2.85907931e+01
613593.0,3582064.0,5.71815861e+01
613600.0,3582057.0,8.56866885e+01
613621.0,3582038.0,1.69285424e+02
613642.0,3582019.0,2.52884160e+02
613662.0,3582000.0,3.36232338e+02
613675.0,3581990.0,3.82235590e+02
613687.0,3581980.0,4.28238841e+02
```

The PEST utility script `mod2obs.exe` is run to extract and interpolate the model-predicted heads at observation locations. The input files for mod2obs.exe were taken from AP-114 Task 7 in CVS. The observed head file has the wells and freshwater heads from Februaru 2010, but is otherwise the same as that used in the model calibration in AP-114. The Python script `merge_observed_modeled_heads.py` (§A-4.9) simply puts the results from `mod2obs.exe` and the original observed heads in a single file together for easier plotting and later analysis.

A similar process to that described so far in this narrative is carried out in a new directory called `pest_02` (beginning line 146 of the driver script). The PEST calibration is carried out there, to keep it separate from the `original_average` simulation. Now the Python script `boundary_types.py` (§A-4.3) is also run, to create a new MODFLOW IBOUND array, where the two different types of boundary conditions are differentiated. This Python script uses the MODFLOW IBOUND array (`init_bnds_orig.inf` first ⅓ of Table 6) and the initial head array (`init_head_orig.mod` middle ⅓ of Table 6) as inputs, and writes a new MODFLOW IBOUND array (`init_bnds.inf` bottom ⅓ of Table 6) with constant-head nodes indicated in red in Figure 1 marked as -2 and other constant-head nodes remaining as -1 as output. The script differentiates between these two types of boundary conditions by checking if the starting head is <1000m. Starting heads >1000m are associated with the constant-head areas to the east of the halite margins (lighter gray areas in Figure 1).

Table 6. Input IBOUND, starting head, and output IBOUND array data corresponding to first row of MODFLOW model

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943 943
943 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944
944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944
944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944
944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944
944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 944 1087 1085
1085 1082 1081 1080 1079 1078 1078 1077 1077 1077 1077 1077 1077 1078 1078 1077 1078 1079 1081
1083 1084 1086 1086 1088 1090 1092 1095 1096 1098 1099 1099 1100 1100 1100 1101 1101 1101 1102
1104 1104 1104 1104 1104 1104 1105 1105 1105 1105 1106 1106 1106 1107 1107 1109 1110 1110 1112
1113 1113 1114 1114 1115 1115 1115 1116 1116 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125
1126 1127 1127 1129 1129 1130 1131 1132 1132 1133 1133 1134 1135 1136 1137 1137 1138 1139 1139
1140 1140 1140 1141 1142 1142 1142 1142 1142 1143 1143 1144 1144 1144 1145 1145 1146 1147 1147
1147 1147 1148 1148 1147 1146 1144 1143 1143 1145 1147 1148 1149 1150 1149 1149 1148 1149 1149
1151 1151 1151 1150 1152 1153 1154 1155
```

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -2
-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Table 6 shows the data corresponding to the northernmost row of the MODFLOW model domain (284 entries long) for the two input files and one output file. In the top IBOUND array, the values are either 0 or -1, indicating either inactive (the region northwest of the no-flow area shown in dark gray in Figure 1) or constant head (both red and light gray cells in Figure 1). The first 284 values from the initial head file (reformatted from scientific notation to integers to facilitate printing) show a jump from approximately 944 (in blue) to >1000 (in red). These same cells are colored in the output, showing how the initial head value is used to distinguish between the two types of constant-head boundaries. MODFLOW treats any cells as constant head which have an IBOUND entry < 0, so both -2 and -1 are the same to MODFLOW, but allow distinguishing between them in the Python script which extrapolates the heads to the boundaries.

The required PEST input files are created by the Python script `create_pest_02_input.py` (§A-4.4). This script writes **1)** the PEST instruction file (`modeled_head.ins`), which shows PEST how to extract the model-predicted heads from the `mod2obs.exe` output; **2)** the PEST template file (`surface_par_params.ptf`), which shows PEST how to write the input file for the surface extrapolation script; **3)** the PEST parameter file (`surface_par_params.par`), which lists the starting parameter values to use when checking the PEST input; **4)** the PEST control file (`bc_adjust_2009ASER.pst`), which has PEST-related parameters, definitions of extrapolation surface parameters, and the observations and weights that PEST is adjusting the model inputs to fit. The observed heads are read as an input file in the PEST borehole sample file format (`meas_head_2009ASER.smp`), and the weights are read in from the input file (`obs_loc_2009ASER.dat`).

PEST runs the "forward model" many times, adjusting inputs and reading the resulting outputs using the instruction and template files created above. The forward model actually consists of a Bash shell script (`run_02_model`) that simply calls a pre-processing Python script `surface_02_extrapolate.py` (§A-4.5), the MODFLOW-2000 executable, the Python script `create_average_NS_residuals.py`, and the PEST utility `mod2obs.exe` as a post-processing step. The script redirects the output of each step to `/dev/null` to minimize screen output while running PEST, since PEST will run the forward model many dozens of times.

The Python script `create_average_NS_residuals.py` takes the output from the PEST utility `mod2obs.exe` and creates a meta-observation that consists of the average residual between measured and model-prediction, only averaged across the northern or southern WIPP wells (the wells in the center of the WIPP site are not included in either group). This was done to minimize cancelation of the errors north (where the model tended to underestimate heads) and south (where the model tended to overestimate heads) of the WIPP. The results of this script are read directly by PEST and incorporated as four additional observations (mean and median errors, both north and south of WIPP).

The pre-processing Python script `surface_02_extrapolate.py` reads the new IBOUND array created in a previous step (with -2 now indicating which constant-head boundaries should be modified), the initial head file used in AP-114 Task 7 (`init_head_orig.mod`), two files listing the relative X and Y coordinates of the model cells (`rel_{x,y}_coord.dat`), and an input file listing the coefficients of the parametric equation used to define the initial head surface. This script then cycles over the elements in the domain, writing the original starting head value if the IBOUND value is -1 or 0, and writing the value corresponding to the parametric equation if the IBOUND value is -2 or 1. Using the parameters corresponding to those used in AP-114 Task 7, the output starting head file should be identical to that used in AP-114 Task 7.

After PEST has converged to the optimum solution for the given observed heads and weights, it runs the forward model one more time with the optimum parameters. The post-processing Python scripts for creating the Surfer ASCII grid file and Surfer blanking file from the MODFLOW and DTRKMF output are run and the results are plotted using additional Python scripts that utilize the plotting and map coordinate projection functionality of the matplotlib library.

These two plotting scripts (`plot-contour-maps.py` and `plot-results-bar-charts.py`) are included in the appendix for completeness, but only draw the figures included in this report, and passed on to WRES for the ASER. These two scripts automate the plotting process and take the place of the Microsoft Excel, USACE Corpscon, and Golden Software Surfer input files that were previously used.

6 Appendix: Files and Script Source Listings

6.1 Input File Listing

bytes	file type	description	file name
2.1K	Python script	average 100 realizations	`average_realizations.py`
2.3K	Python script	distinguish different BC types	`boundary_types.py`
6.6K	Bash script	main routine: checkout files,run MODFLOW run PEST, call Python scripts	`checkout_average_run_modflow.sh`
809	Python script	convert DTRKMF IJ output to Surfer X,Y blanking format	`convert_dtrkmf_output_for_surfer.py`
3.2K	Python script	create PEST input files from observed data	`create_pest_02_input.py`
48	input listing	responses to DTRKMF prompts	`dtrkmf.in`
4.2K	Python script	convert MODFLOW binary output to Surfer ASCII grid format	`head_bin2ascii.py`
1.1K	input	listing of 100 realizations from CVS	`keepers`
1.4K	input	observed February 2010 heads in `mod2obs.exe` bore sample file format	`meas_head_2010ASER.smp`
1.2K	Python script	paste observed head and model-generated heads into one file	`merge_observed_modeled_heads.py`
76	file listing	files needed to run `mod2obs.exe`	`mod2obs_files.dat`
138	input listing	responses to `mod2obs.exe` prompts	`mod2obs_head.in`
372	file listing	files needed to run MODFLOW	`modflow_files.dat`
401	input	listing of wells and geographic groupings	`obs_loc_2010ASER.dat`
215	file listing	files needed to run PEST	`pest_02_files.dat`
2.3M	input	relative coordinate $1 \leq x \leq 1$	`rel_x_coord.dat`
2.3M	input	relative coordinate $1 \leq y \leq 1$	`rel_y_coord.dat`
389	Bash script	PEST model: execute MODFLOW and do pre- and post-processing	`run_02_model`
26	input	`mod2obs.exe` input file	`settings.fig`
47	input	`mod2obs.exe` input file	`spec_domain.spc`
1.7K	input	`mod2obs.exe` input file	`spec_wells.crd`
2.7K	Python script	compute starting head from parameter and coordinate inputs	`surface_02_extrapolate.py`
506	input	DTRKMF input file	`wippctrl.inp`
5.6K	Python script	plot contour map figures	`plot-contour-maps.py`
6.7K	Python script	plot bar and scatter figures	`plot-results-bar-charts.py`
90	plotting data	UTM coordinates of ASER map area	`ASER_boundary.csv`
9.2K	plotting data	UTM coordinates of MODFLOW model area	`total_boundary.dat`
6.7K	plotting data	UTM coordinates of WIPP LWB	`wipp_boundary.csv`

Table 1: Listing of Input Files

6.2 Output File Listing

bytes	file type	description	file name
19K	DTRKMF output	particle track results	`dtrk.out`
16K	DTRKMF output	particle track debug	`dtrk.dbg`
2.0K	script output	heads at well locations	`modeled_vs_observed_head_pest_02.txt`
1.1M	script output	formatted MODFLOW heads	`modeled_head_pest_02.grd`
5.3K	script output	formatted DTRKMF particle	`dtrk_output_pest_02.bln`
16K	PEST output	matrix condition numbers	`bc_adjust_2010ASER.cnd`
2.7K	PEST output	binary intermediate file	`bc_adjust_2010ASER.drf`
7.4K	PEST output	binary intermediate file	`bc_adjust_2010ASER.jac`
7.5K	PEST output	binary intermediate file	`bc_adjust_2010ASER.jco`
9.9K	PEST output	binary intermediate file	`bc_adjust_2010ASER.jst`
3.8K	PEST output	parameter statistical matrices	`bc_adjust_2010ASER.mtt`
477	PEST output	parameter file	`bc_adjust_2010ASER.par`
62K	PEST output	optimization record	`bc_adjust_2010ASER.rec`
4.6K	PEST output	model outputs for last iteration	`bc_adjust_2010ASER.rei`
8.4K	PEST output	summary of residuals	`bc_adjust_2010ASER.res`
28	PEST output	binary restart file	`bc_adjust_2010ASER.rst`
24K	PEST output	relative parameter sensitivities	`bc_adjust_2010ASER.sen`
4.0K	PEST output	absolute parameter sensitivities	`bc_adjust_2010ASER.seo`
213K	png image	matplotlib plot (Fig. 2)	`aser-area-contour-map.png`
223K	png image	matplotlib plot (Fig. 3)	`large-area-contour-map.png`
33K	png image	matplotlib plot (Fig. 5)	`model-error-histogram.png`
55K	png image	matplotlib plot (Fig. 6)	`model-error-residuals.png`
93K	png image	matplotlib plot (Fig. 4)	`scatter_pest_02.png`

Table 2: Listing of Output Files

6.3 Individual Script Listings

6.3.1 Bash shell script `checkout_average_run_modflow.sh`

```
#!/bin/bash

# set -o xtrace # for debugging
set -o nounset # quit if using an un-initialized variable
set -o errexit # exit on non-zero error status of sub-command

# this script makes the following directory substructure
#
#  current_dir \------ Outputs  (calibrated parameter fields - INPUTS)
#               \----- Inputs   (other modflow files - INPUTS)
#                \---- original_average  (foward sim using average fields)
#                 \--- bin      (MODFLOW and DTRKMF binaries)
#                  \-- pest_0?  (pest-adjusted results)


echo " ^^^^^^^^^^^^^^^^^^^^ "
echo " checking out T fields"
echo " ^^^^^^^^^^^^^^^^^^^^ "

# these will checkout the calibrated parameter-field data into subdirectories
# checkout things that are different for each of the 100 realiztaions
for d in `cat keepers`
do
  cvs -d /nfs/data/CVSLIB/Tfields checkout Outputs/${d}/modeled_{K,A,R,S}_field.mod
done

# checkout MODFLOW input files that are constant for across all realizations
cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/data/elev_{top,bot}.mod
cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/data/init_{bnds.inf,head.mod}
cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/modflow/mf2k_culebra.{lmg,lpf}
cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/modflow/mf2k_head.{ba6,nam,oc,dis,rch}

# modify the path of "updated" T-fields, so they are all at the
# same level in the directory structure (simplifying scripts elsewhere)

if [ -a keepers_short ]
then
    rm keepers_short
fi
touch keepers_short

for d in `cat keepers`
do
  bn=`basename ${d}`
  # test whether it is a compound path
  if [ ${d} != ${bn} ]
      then
      dn=`dirname ${d}`
      mv ./Outputs/${d} ./Outputs/

      # put an empty file in the directory to indicate
      # what the directory was previously named
      touch ./Outputs/${bn}/${dn}
  fi

  # create a keepers list without directories
  echo ${bn} >> keepers_short
done

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

```

```
echo " ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ "
echo " perform averaging across all realizations "
echo " ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ "

python average_realizations.py

# checkout MODFLOW / DTRKMF executables
cvs -d /nfs/data/CVSLIB/MODFLOW2K checkout bin/mf2k/mf2k_1.6.release
cvs -d /nfs/data/CVSLIB/MODFLOW2K checkout bin/dtrkmf/dtrkmf_v0100

# check out pest and obs2mod binaries
cd bin
cvs -d /nfs/data/CVSLIB/PEST checkout Builds/Linux/pest.exe
cvs -d /nfs/data/CVSLIB/PEST checkout Builds/Linux/mod2obs.exe
cd ..

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

echo " ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ "
echo " setup copies of files constant between all realizations "
echo " ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ "

# directory for putting original base-case results in
od=original_average

if [ -d ${od} ]
then
    echo ${od}" directory exists: removing and re-creating"
    rm -rf ${od}
fi

mkdir ${od}
cd ${od}
echo `pwd`

# link to unchanged input files
for file in `cat ../modflow_files.dat`
do
  ln -sf ${file} .
done

# link to averaged files computed in previous step
for f in {A,R,K,S}
do
  ln -sf ../modeled_${f}_field.avg ./modeled_${f}_field.mod
done

ln -sf elev_top.mod fort.33
ln -sf elev_bot.mod fort.34

echo "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"
echo " run original MODFLOW and DTRKMF and export results for plotting"
echo "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"

# run MODFLOW, producing average head and CCF
../bin/mf2k/mf2k_1.6.release mf2k_head.nam

# run DTRKMF, producing particle track (from ccf)
../bin/dtrkmf/dtrkmf_v0100 <dtrkmf.in

# convert binary MODFLOW head output to Surfer ascii grid file format
ln -sf ../head_bin2ascii.py .
python head_bin2ascii.py
mv modeled_head_asciihed.grd modeled_head_${od}.grd
```

```

# convert DTRKMF output from cells to X,Y and
# save in Surfer blanking file format
ln -sf ../convert_dtrkmf_output_for_surfer.py .
python convert_dtrkmf_output_for_surfer.py
mv dtrk_output.bln dtrk_output_${od}.bln

# extract head results at well locations and merge with observed
# head file for easy scatter plotting in Excel (tab delimited)
for file in `cat ../mod2obs_files.dat`
do
  ln -sf ${file} .
done

ln -sf ../meas_head_2010ASER.smp .
ln -sf ../obs_loc_2010ASER.dat .
../bin/Builds/Linux/mod2obs.exe <mod2obs_head.in
ln -sf ../merge_observed_modeled_heads.py
python merge_observed_modeled_heads.py
mv both_heads.smp modeled_vs_observed_head_${od}.txt


# go back down into root directory
cd ..
echo `pwd`

echo "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"
echo " setup and run PEST to optimize parametric surface to set BC "
echo "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"

for p in pest_02
do

  if [ -d ${p} ]
      then
      echo ${p}" directory exists: removing and re-creating"
      rm -rf ${p}
  fi

  mkdir ${p}
  cd ${p}
  echo `pwd`

  # link to unchanged input files
  for file in `cat ../modflow_files.dat`
    do
    ln -sf ${file} .
  done

  # link to averaged files computed in previous step
  for f in {A,R,K,S}
    do
    ln -sf ../modeled_${f}_field.avg ./modeled_${f}_field.mod
  done

  # link to mod2obs files (needed for pest)
  for file in `cat ../mod2obs_files.dat`
    do
    ln -sf ${file} .
  done

  # link to pest files
  for file in `cat ../${p}_files.dat`
    do
    ln -s ${file} .
  done

  # rename 'original' versions of files to be modified by pest
  rm init_head.mod
  ln -sf ../Inputs/data/init_head.mod ./init_head_orig.mod
  rm init_bnds.inf
  ln -sf ../Inputs/data/init_bnds.inf ./init_bnds_orig.inf

  # create new ibound array for easier modification during PEST
  # optimization iterations
  python boundary_types.py

  # create the necessary input files from observations
  python create_${p}_input.py

  # run pest
  ../bin/Builds/Linux/pest.exe bc_adjust_2010ASER

  # last output files should be best run
  # extract all the stuff from that output
  ##########################################

  ln -sf elev_top.mod fort.33
  ln -sf elev_bot.mod fort.34

  ../bin/dtrkmf/dtrkmf_v0100 <dtrkmf.in

  ln -sf ../head_bin2ascii.py .
  python head_bin2ascii.py
  mv modeled_head_asciihed.grd modeled_head_${p}.grd

  ln -sf ../convert_dtrkmf_output_for_surfer.py .
  python convert_dtrkmf_output_for_surfer.py
  mv dtrk_output.bln dtrk_output_${p}.bln

  for file in `cat ../mod2obs_files.dat`
  do
    ln -sf ${file} .
  done

  ../bin/Builds/Linux/mod2obs.exe <mod2obs_head.in
  ln -sf ../merge_observed_modeled_heads.py
  python merge_observed_modeled_heads.py
  mv both_heads.smp modeled_vs_observed_head_${p}.txt

  cd ..
done
```

6.3.2 Python script average_realizations.py

```python
from math import log10,pow

nrow = 307
ncol = 284
nel = nrow*ncol
nfr = 100  # number of fields (realizations)
nft = 4    # number of field types

debug = True  # set to True to get output described in RunControl narrative

def floatload(filename):
    """Reads file (a list of strings, one per row) into a list of strings."""
    f = open(filename,'r')
    m = [float(line.rstrip()) for line in f]
    f.close()
    return m

types = ['K','A','R','S']

# get list of 100 best calibrated fields
flist = open('keepers_short','r')
runs = flist.read().strip().split('\n')
flist.close()

# initialize to help speed lists up a bit
# nfr (100) realizations of each
fields = []
for i in xrange(nft):
    fields.append([None]*nfr)
    for i in xrange(nfr):
        # each realization being nel (87188) elements
        fields[-1][i] = [None]*nel

# read in all realizations
print 'reading ...'
for i,run in enumerate(runs):
    print i,run
    for j,t in enumerate(types):
        fields[j][i][0:nel] = floatload('Outputs/'+run+'/modeled_'+t+'_field.mod')

# save file with one cell from each realization for checking in Excel
if debug:
    print 'writing debugging output for checking'
    fd = open('parameter_representative_values.txt','w')
    fd.write('%s %18s %18s %18s %18s\n'%
             ('rzn',types[0],types[1],types[2],types[3]))
    for i,run in enumerate(runs):
        fd.write('%s %.14e %.14e %.14e %.14e\n' %
                 (run,fields[0][i][-1],fields[1][i][-1],
                  fields[2][i][159*284],fields[3][i][159*284]))
    fd.close()

# open up files for writing
fh = []
for t in types:
    fh.append(open('modeled_'+ t +'_field.avg','w'))

# transpose fields to allow slicing across realizations, rather than across cells
for j in range(len(types)):
    fields[j] = zip(*(fields[j]))

print 'writing ...'
# do averaging across 100 realizations
```

```
for i in xrange(nel):
    if i%10000 == 0:
        print i
    for h,d in zip(fh,fields):
        h.write('%18.11e\n' % pow(10.0,sum(map(log10,d[i]))/nfr) )

for h in fh:
    h.close()
```

6.3.3 Python script boundary_types.py

```python
# this python script computes some summary residuals
# based on the concept of "north of WIPP" and "south of WIPP"
# to get PEST to honor the areas outside the steep gradient
# across the site.

nx = 284       # number columns in model grid
ny = 307       # number rows
nel = nx*ny

def intload(filename):
    """Reads file (a 2D integer array) as a list of lists.
    Outer list is rows, inner lists are columns."""
    f = open(filename,'r')
    m = [[int(v) for v in line.rstrip().split()] for line in f]
    f.close()
    return m

def intsave(filename,m):
    """Writes file as a list of lists as a 2D integer array, format '%3i'.
    Outer list is rows, inner lists are columns."""
    f = open(filename,'w')
    for row in m:
        f.write(' '.join(['%2i' % col for col in row]) + '\n')
    f.close()

def floatload(filename):
    """Reads file (a list of real numbers, one number each row) into a list of floats."""
    f = open(filename,'r')
    m = [float(line.rstrip()) for line in f]
    f.close()
    return m

def reshapev2m(v):
    """Reshape a vector that was previously reshaped in C-major order from a matrix,
    back into a matrix (here a list of lists)."""
    m = [None]*ny
    for i,(lo,hi) in enumerate(zip(xrange(0, nel-nx+1, nx), xrange(nx, nel+1, nx))):
        m[i] = v[lo:hi]
    return m

#############################################

# read in original MODFLOW IBOUND array (only 0,1, and -1)
ibound = intload('init_bnds_orig.inf')

# read in initial heads
h = reshapev2m(floatload('init_head_orig.mod'))

# discriminate between two types of constant head boundaries
# -1) CH, where value > 1000 (area east of halite margin)
# -2) CH, where value < 1000 (single row/column of cells along edge of domain

for i,row in enumerate(ibound):
    for j,val in enumerate(row):
        # is this constant head and is starting head less than 1000m ?
        if ibound[i][j] == -1 and h[i][j] < 1000.0:
            ibound[i][j] = -2

# save new IBOUND array that allows easy discrimination between types in python script durin
# PEST optimization runs, and is still handled the same by MODFLOW
# since all ibound values < 0 are treated as constant head.
intsave('init_bnds.inf',ibound)
```

6.3.4 Python script create_pest_02_input.py

```python
# this python script computes some summary residuals
# based on the concept of "north of WIPP" and "south of WIPP"
# to get PEST to honor the areas outside the steep gradient
# across the site.

prefix = '2010ASER'

###########################################################
## pest instruction file reads output from mod2obs
fin = open('meas_head_%s.smp' % prefix,'r')

# each well is a [name,head] pair
wells = [[line.split()[0],line.split()[3]] for line in fin]
fin.close()

fout = open('modeled_head.ins','w')
fout.write('pif @\n')
for i,well in enumerate(wells):
        fout.write("l1 [%s]39:46\n" % well[0])
fout.close()

# exponential surface used to set initial head everywhere
# except east of the halite margins, where the land surface is used.
# initial guesses come from AP-114 Task report
params = [928.0, 8.0, 1.2, 1.0, 1.0, -1.0, 0.5]
pnames = ['a',   'b', 'c', 'd', 'e', 'f', 'exp']

fout = open('avg_NS_res.ins','w')
fout.write("""pif @
l1 [medianN]1:16
l1 [medianS]1:16
l1 [meanN]1:16
l1 [meanS]1:16
""")
fout.close()


##################################
## pest template file
ftmp = open('surface_par_params.ptf','w')
ftmp.write('ptf @\n')
for n in pnames:
        ftmp.write('@      %s        @\n' % n)
ftmp.close()


####################
## pest parameter file

fpar = open('surface_par_params.par','w')
fpar.write('double  point\n')
for n,p in zip(pnames,params):
    fpar.write('%s  %.2f  1.0  0.0\n' % (n,p))
fpar.close()


#####################
## pest control file

f = open('bc_adjust_%s.pst' % prefix,'w')

f.write("""pcf
* control data
```

```
restart estimation
%i %i 1 0 2
1 2 double point 1 0 0
5.0 2.0 0.4 0.001 10
3.0 3.0 1.0E-3
0.1
30 0.001 4 4 0.0001 4
1 1 1
* parameter groups
bc relative 0.005 0.0001 switch 2.0 parabolic
""" % (len(params),len(wells)+4))

f.write('* parameter data\n')
for n,p in zip(pnames,params):
        if p > 0:
                f.write('%s  none  relative  %.3f  %.3f  %.3f  bc  1.0  0.0  1\n' %
                        (n, p, -2.0*p, 3.0*p))
        else:
                f.write('%s  none  relative  %.3f  %.3f  %.3f  bc  1.0  0.0  1\n' %
                        (n, p, 3.0*p, -2.0*p))

f.write("""* observation groups
ss_head
avg_head
* observation data
""")

## read in observation weighting group definitions
fin = open('obs_loc_%s.dat' % prefix,'r')
location = [line.rstrip().split()[1] for line in fin]
fin.close()

weights = []

for l in location:
    # inside LWB
    if l == '0':
        weights.append(2.5)
    # near LWB
    if l == '1':
        weights.append(1.0)
    # distant to LWB
    if l == '2':
        weights.append(0.4)
    if l == '99':
        weights.append(0.01) # AEC-7


for name,head,w in zip(zip(*wells)[0],zip(*wells)[1],weights):
    f.write('%s  %s  %.3f   ss_head\n' % (name,head,w))

# one fewer N observation (WIPP-25 removed), there were 13
# there are 12 N observations in the average and 11 S, therefore
# split the weight between the mean and median
f.write("""medianN  0.0  18.0  avg_head
medianS  0.0  16.5  avg_head
meanN    0.0  18.0  avg_head
meanS    0.0  16.5  avg_head
""")

f.write("""* model command line
./run_02_model
* model input/output
surface_par_params.ptf surface_par_params.in
```

```
modeled_head.ins modeled_head.smp
avg_NS_res.ins avg_NS_res.smp
""")
f.close()
```

6.3.5 Python script surface_02_extrapolate.py

```python
# this python script is used in the analysis associated
# with SP 9-9: "Analysis Report for Preparation of 2010 Culebra Potentiometric
# Surface Contour Maps."
# by Kris Kuhlman (2011)

from itertools import chain
from math import sqrt

def matload(filename):
    """Reads file (a 2D string array) as a list of lists.
    Outer list is rows, inner lists are columns."""
    f = open(filename,'r')
    m = [line.rstrip().split() for line in f]
    f.close()
    return m

def floatload(filename):
    """Reads file (a list of real numbers, one number each
    row) into a list of floats."""
    f = open(filename,'r')
    m = [float(line.rstrip()) for line in f]
    f.close()
    return m

def reshapem2v(m):
    """Reshapes a rectangular matrix into a vector in same
    fashion as numpy.reshape().
    which is C-major order"""
    return list(chain(*m))

def sign(x):
    """ sign function"""
    if x<0:
        return -1
    elif x>0:
        return +1
    else:
        return 0

##########################################

# read in modified IBOUND array, with the cells to modify set to -2
ibound = reshapem2v(matload('init_bnds.inf'))

h = floatload('init_head_orig.mod')

# these are relative coordinates, -1 <= x,y < +1
x = floatload('rel_x_coord.dat')
y = floatload('rel_y_coord.dat')

# unpack surface parameters (one per line)
# z = A + B*(y + D*sign(y)*sqrt(abs(y)))+C*(E*x**3 - F*x**2 - x)

finput = open('surface_par_params.in','r')
try:
    a,b,c,d,e,f,exp = [float(line.rstrip()) for line in finput]
except ValueError:
    # python doesn't like 'D' in 1.2D-4 notation used by PEST sometimes.
    finput.seek(0)
    lines =  [line.rstrip() for line in finput]
    for i in range(len(lines)):
        lines[i] = lines[i].replace('D','E')
    a,b,c,d,e,f,exp = [float(line) for line in lines]
```

```

finput.close()

# file to output initial/boundary head for MODFLOW model
fout = open('init_head.mod','w')
for i in xrange(len(ibound)):
    if ibound[i] == '-2' or ibound[i] == '1':
        # apply exponential surface to active cells (ibound=1) -> starting guess
        # and non-geologic boundary conditions (ibound=-2) -> constant head value
        if y[i] == 0:
            fout.write('%.7e \n' % (a + c*(e*x[i]**3 + f*x[i]**2 - x[i])))
        else:
            fout.write('%.7e \n' % (a + b*(y[i] + d*sign(y[i])*abs(y[i])**exp) +
                                         c*(e*x[i]**3 + f*x[i]**2 - x[i])))
    else:
        # use land surface at constant head east of halite boundary
        # ibound=0 doesn't matter (inactive)
        fout.write('%.7e\n' % h[i])

fout.close()
```

6.3.6 Bash shell script `run_02_model`

```bash
#!/bin/bash

#set -o xtrace

#echo 'step 1: surface extrapolate'
python surface_02_extrapolate.py

# run modflow
#echo 'step 2: run modflow'
../bin/mf2k/mf2k_1.6.release mf2k_head.nam  >/dev/null

# run mod2obs
#echo 'step 3: extract observations'
../bin/Builds/Linux/mod2obs.exe < mod2obs_head.in  >/dev/null

# create meta-observations of N vs. S
python create_average_NS_residuals.py
```

6.3.7 Python script head_bin2ascii.py

```python
# this python script is used in the analysis associated
# with SP 9-9: "Analysis Report for Preparation of 2010 Culebra Potentiometric
# Surface Contour Maps."
# by Kris Kuhlman (2011)

import struct
from sys import argv,exit

class FortranFile(file):
    """ modified from May 2007 Enthought-dev mailing
    list post by Neil Martinsen-Burrell"""

    def __init__(self,fname, mode='r', buf=0):
        file.__init__(self, fname, mode, buf)
        self.ENDIAN = '<'  # little endian
        self.di = 4  # default integer (could be 8 on 64-bit platforms)

    def readReals(self, prec='f'):
        """Read in an array of reals (default single
        precision) with error checking"""
        # read header (length of record)
        l = struct.unpack(self.ENDIAN+'i',self.read(self.di))[0]
        data_str = self.read(l)
        len_real = struct.calcsize(prec)
        if l % len_real != 0:
            raise IOError('Error reading array of reals from data file')
        num = l/len_real
        reals = struct.unpack(self.ENDIAN+str(num)+prec,data_str)
        # check footer
        if struct.unpack(self.ENDIAN+'i',self.read(self.di))[0] != l:
            raise IOError('Error reading array of reals from data file')
        return list(reals)

    def readInts(self):
        """Read in an array of integers with error checking"""
        l = struct.unpack('i',self.read(self.di))[0]
        data_str = self.read(l)
        len_int = struct.calcsize('i')
        if l % len_int != 0:
            raise IOError('Error reading array of integers from data file')
        num = l/len_int
        ints = struct.unpack(str(num)+'i',data_str)
        if struct.unpack(self.ENDIAN+'i',self.read(self.di))[0] != l:
            raise IOError('Error reading array of integers from data file')
        return list(ints)

    def readRecord(self):
        """Read a single fortran record (potentially mixed
        reals and ints)"""
        dat = self.read(self.di)
        if len(dat) == 0:
            raise IOError('Empy record header')
        l = struct.unpack(self.ENDIAN+'i',dat)[0]
        data_str = self.read(l)
        if len(data_str) != l:
            raise IOError('Didn''t read enough data')
        check = self.read(self.di)
        if len(check) != 4:
            raise IOError('Didn''t read enough data')
        if struct.unpack(self.ENDIAN+'i',check)[0] != l:
            raise IOError('Error reading record from data file')
        return data_str

```

```python
def reshapev2m(v,nx,ny):
    """Reshape a vector that was previously reshaped in C-major order from a matrix,
    back into a C-major order matrix (here a list of lists)."""
    m = [None]*ny
    n = nx*ny
    for i,(lo,hi) in enumerate(zip(xrange(0, n-nx+1, nx), xrange(nx, n+1, nx))):
        m[i] = v[lo:hi]
    return m

def floatmatsave(filehandle,m):
    """Writes array to open filehandle, format '568%e12.5'.
    Outer list is rows, inner lists are columns."""

    for row in m:
        f.write(''.join([' %12.5e' % col for col in row]) + '\n')

# open file and set endian-ness
try:
    infn,outfn = argv[1:3]
except:
    print '2 command-line arguments not given, using default in/out filenames'
    infn = 'modeled_head.bin'
    outfn = 'modeled_head_asciihed.grd'

ff = FortranFile(infn)

# currently this assumes a single-layer MODFLOW model
# (or at least only one layer of output)

# format of MODFLOW header in binary layer array
fmt = '<2i2f16s3i'
# little endian, 2 integers, 2 floats,
#    16-character string (4 element array of 4-byte strings), 3 integers

while True:
    try:
        # read in header
        h = ff.readRecord()

    except IOError:
        # exit while loop
        break

    else:
        # unpack header
        kstp,kper,pertim,totim,text,ncol,nrow,ilay = struct.unpack(fmt,h)

        # print status/confirmation to terminal
        print kstp,kper,pertim,totim,text,ncol,nrow,ilay

        h = ff.readReals()

ff.close()

xmin, xmax = (601700.0,630000.0)
ymin, ymax = (3566500.0,3597100.0)
hmin = min(h)
hmax = max(h)

# write output in Surfer ASCII grid format
f = open(outfn,'w')
f.write("""DSAA
%i %i
%.1f %.1f
%.1f %.1f
%.8e %.8e
""" %(ncol,nrow,xmin,xmax,ymin,ymax,hmin,hmax) )
hmat = reshapev2m(h,ncol,nrow)

# MODFLOW starts data in upper-left corner
# Surfer expects data starting in lower-left corner
# flip array in row direction

floatmatsave(f,hmat[::-1])
f.close()
```

6.3.8 Python script merge_observed_modeled_heads.py

```
# this python script is used in the analysis associated
# with SP 9-9: "Analysis Report for Preparation of 2010 Culebra Potentiometric
# Surface Contour Maps."
# by Kris Kuhlman (2011)

fobs = open('meas_head_2010ASER.smp','r') # measured head
fmod = open('modeled_head.smp','r')       # modeled head
fwgt = open('obs_loc_2010ASER.dat','r')   # weights
fdb = open('spec_wells.crd','r')          # x/y coordinates

fout = open('both_heads.smp','w')         # resulting file

# read in list of x/y coordinates, key by well name
wells = {}
for line in fdb:
    well,x,y = line.split()[0:3] # ignore last column
    wells[well.upper()] = [x,y]
fdb.close()

fout.write('\t'.join(['#NAME','UTM-NAD27-X','UTM-NAD27-Y',
                      'OBSERVED','MODELED','OBS-MOD','WEIGHT'])+'\n')

for sobs,smod,w in zip(fobs,fmod,fwgt):
    obs = float(sobs.split()[3])
    mod = float(smod.split()[3])
    name = sobs.split()[0].upper()
    fout.write('\t'.join([name,wells[name][0],wells[name][1],
                          str(obs),str(mod),str(obs-mod),
                          w.rstrip().split()[1]])+'\n')

fobs.close()
fmod.close()
fwgt.close()
fout.close()
```

6.3.9 Python script `convert_dtrkmf_output_for_surfer.py`

```python
# this python script is used in the analysis associated
# with SP 9-9: "Analysis Report for Preparation of 2010 Culebra Potentiometric
# Surface Contour Maps."
# by Kris Kuhlman (2011)

# grid origin for dtrkmf cell -> x,y conversion
x0 = 601700.0
y0 = 3597100.0

dx = 100.0
dy = 100.0

fout = open('dtrk_output.bln','w')

# read in all results for saving particle tracks
fin = open('dtrk.out','r')
results = [l.split() for l in fin.readlines()[1:]]
fin.close()

npts = len(results)

# write Surfer blanking file header
fout.write('%i,1\n' % npts)

# write x,y location and time
for pt in results:
    x = float(pt[1])*dx + x0
    y = y0 - float(pt[2])*dy
    t = float(pt[0])/7.75*4.0  # convert to 4m Cuelbra thickness
    fout.write('%.1f,%.1f,%.8e\n' % (x,y,t))

fout.close()
```

6.3.10 Python script `plot-results-bar-charts.py`

This script is not run on the QA linux cluster, `alice.sandia.gov`. This script is run on a desktop PC, but is only used to create figures for the analysis report. This script is only included here for completeness.

```
# this python script is used to plot results associated
# with SP 9-9: "Analysis Report for Preparation of 2010 Culebra Potentiometric
# Surface Contour Maps."
# by Kris Kuhlman (2011)

import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

fname = 'modeled_vs_observed_head_pest_02.txt'

M2FT = 0.3048

# load in observed, modeled, obs-mod, (all in meters)
res = np.loadtxt(fname,skiprows=1,usecols=(3,4,5))
# load in weights
weights = np.loadtxt(fname,skiprows=1,usecols=(6,),dtype='int')
# load in names
names = np.loadtxt(fname,skiprows=1,usecols=(0,),dtype='|S6')

## checking locations / zones
# ********************************************
wipp = np.loadtxt('wipp_boundary.dat')
x,y = np.loadtxt(fname,skiprows=1,usecols=(1,2),unpack=True)

fig = plt.figure(2,figsize=(18,12))
ax1 = fig.add_subplot(121)
ax1.plot(x,y,'k*') # wells
ax1.plot(wipp[:,0],wipp[:,1],'r-') # WIPP LWB
buff = np.loadtxt('wipp_boundary.dat')
buff[1:3,0] -= 3000.0
buff[0,0] += 3000.0
buff[3:,0] += 3000.0
buff[2:4,1] -= 3000.0
buff[0:2,1] += 3000.0
buff[-1,1] += 3000.0
ax1.plot(buff[:,0],buff[:,1],'g--') # WIPP LWB+3km
for xv,yv,n,w in zip(x,y,names,weights):
    plt.annotate('%s %i'%(n,w),xy=(xv,yv),fontsize=8)
plt.axis('image')
ax1.set_xlim([x.min()-1000,x.max()+1000])
ax1.set_ylim([y.min()-1000,y.max()+1000])
ax2 = fig.add_subplot(122)
ax2.plot(x,y,'k*') # wells
ax2.plot(wipp[:,0],wipp[:,1],'r-') # WIPP LWB
ax2.plot(buff[:,0],buff[:,1],'g--') # WIPP LWB+3km
for xv,yv,n,w in zip(x,y,names,weights):
    plt.annotate('%s %i'%(n,w),xy=(xv,yv),fontsize=8)
plt.axis('image')
ax2.set_xlim([wipp[:,0].min()-100,wipp[:,0].max()+100])
ax2.set_ylim([wipp[:,1].min()-100,wipp[:,1].max()+100])
plt.savefig('check-well-weights.png')

# convert lengths to meters
res /= M2FT

# create the histogram of residuals for ASER
# ***************************************

# -10,-9,...8,9,10
```

```
bins = np.arange(-10,11)
rectfig = (15,7)
squarefig = (8.5,8.5)

fig = plt.figure(1,figsize=rectfig)
ax = fig.add_subplot(111)
# all the data, all but distant wells
ax.hist([res[weights<2,2],res[:,2]],bins=bins,range=(-10.0,10.0),
         rwidth=0.75,align='mid',
         color=['red','blue'],
         label=['Inside LWB & <3km from WIPP LWB','All wells'])
ax.set_xlabel('Measured-Modeled (ft)')
ax.set_ylabel('Frequency')
ax.set_xticks(bins)
ax.set_ylim([0,10])
ax.set_yticks(np.arange(0,10,2))
plt.grid()
ax.yaxis.grid(True,which='major')
ax.xaxis.grid(False)
plt.legend(loc='upper left')
plt.annotate('AEC-7 @ %.1f'%res[0,2],xy=(-9.75,5.0),xytext=(-8.5,5.0),
                arrowprops={'arrowstyle':'->'},fontsize=16)
plt.savefig('model-error-histogram.png')
plt.close(1)

# create bar chart plot of individual residual for ASER
# ****************************************

# separate wells into groups
resin =    res[weights==0,2]
resnear = res[weights==1,2]
resfar =  res[weights==2,2]

nin = resin.size
nnear = resnear.size
nfar = resfar.size

# separate names into groups
namin =    names[weights==0]
namnear = names[weights==1]
namfar =  names[weights==2]

# get indices that sort vectors
ordin = np.argsort(namin)
ordnear = np.argsort(namnear)
ordfar = np.argsort(namfar)

# put vectors back together (groups adjacent and sorted inside each group)
resagg = np.concatenate((resin[ordin],resnear[ordnear],resfar[ordfar]),axis=0)
namagg = np.concatenate((namin[ordin],namnear[ordnear],namfar[ordfar]),axis=0)

fig = plt.figure(1,figsize=rectfig)
ax = fig.add_subplot(111)

wid = 0.6
shift = 0.5 - wid/2.0
ab = np.arange(res.shape[0])

ax.bar(left=ab+shift,height=resagg,width=0.6,bottom=0.0,color='gray')
ax.set_ylim([-15.0,15.0])
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
plt.xticks(ab+wid,namagg,rotation=90)
```

```
# vertical lines dividing groups
ax.axvline(x=nin,color='black',linestyle='dashed')
ax.axvline(x=nin+nnear,color='black',linestyle='dashed')
ax.axhline(y=0,color='black',linestyle='solid')
ax.axhline(y=-15,color='black',linestyle='dotted')
plt.grid()
ax.yaxis.grid(True,which='major')
ax.xaxis.grid(False)
ax.set_xlim([0,res.shape[0]])

plt.annotate('',xy=(0.0,12.0),xycoords='data',
             xytext=(nin,12.0),textcoords='data',
             arrowprops={'arrowstyle':'<->'})
plt.annotate('inside WIPP LWB',xy=(nin/3.0,12.5),xycoords='data')

plt.annotate('',xy=(nin,12.0),xycoords='data',
             xytext=(nin+nnear,12.0),textcoords='data',
             arrowprops={'arrowstyle':'<->'})
plt.annotate('<3km WIPP LWB',xy=(nin+nnear/3.0,12.5),xycoords='data')

plt.annotate('',xy=(nin+nnear,12.0),xycoords='data',
             xytext=(nin+nnear+nfar,12.0),textcoords='data',
             arrowprops={'arrowstyle':'<->'})
plt.annotate('>3km WIPP LWB',xy=(nin+nnear+nfar/3.0,12.5),xycoords='data')

plt.annotate('AEC-7 @ %.1f'%res[0,2],xy=(nin+nnear+1.0,-14.5),xycoords='data')

ax.set_ylabel('Measured-Modeled (ft)')
plt.savefig('model-error-residuals.png')
plt.close(1)


# create scatter plot of measured vs. modeled
# ****************************************
m = 1.0/M2FT
sr = [2980,3120]

print 'modeled-vs-measured correlation coefficients\n', \
      'all data:',   np.corrcoef(res[:,0],          res[:,1])**2,'\n', \
      'inside WIPP:',np.corrcoef(res[weights==0,0],res[weights==0,1])**2,'\n', \
      'inside 3km:', np.corrcoef(res[weights<2,0], res[weights<2,1])**2


fig = plt.figure(1,figsize=squarefig)
ax = fig.add_subplot(111)
ax.plot(res[weights==0,0],res[weights==0,1],color='red',markersize=10,
        marker='+',linestyle='none',label='Inside LWB')
ax.plot(res[weights==1,0],res[weights==1,1],color='green',markersize=10,
        marker='x',linestyle='none',label='< 3km From LWB')
ax.plot(res[weights==2,0],res[weights==2,1],color='blue',markersize=10,
        marker='*',linestyle='none',label='distant')
ax.plot(sr,sr,'k-',label='$45^{\\degree}$ Perfect Fit')
ax.plot([sr[0],sr[1]],[sr[0]+m,sr[1]+m],'g-',linewidth=0.5,label='$\\pm$ 1m Misfit')
ax.plot([sr[0],sr[1]],[sr[0]-m,sr[1]-m],'g-',linewidth=0.5,label='__nolegend__')
ax.set_xticks(np.linspace(sr[0],sr[1],8))
ax.set_yticks(np.linspace(sr[0],sr[1],8))
ax.set_xlim(sr)
ax.set_ylim(sr)
plt.minorticks_on()
plt.legend(loc='lower right',scatterpoints=1,numpoints=1)
plt.grid()
for j,lab in enumerate(names):
    if res[j,2] < -1.5*m:
        # plot labels to left of value far above 45-degree line
        plt.annotate(lab,xy=(res[j,0],res[j,1]),
                     xytext=(res[j,0]-(2.9*len(lab)),res[j,1]-2.0),fontsize=14)
    elif res[j,2] > 1.5*m:
        # plot labels to right of value far below 45-degree line
        plt.annotate(lab,xy=(res[j,0],res[j,1]),
                     xytext=(res[j,0]+2.0,res[j,1]-2.0),fontsize=14)
ax.set_xlabel('Observed Freshwater Head (ft AMSL)')
ax.set_ylabel('Modeled Freshwater Head (ft AMSL)')
plt.savefig('scatter_pest_02.png')
```

6.3.11 Python script `plot-contour-maps.py`

This script is not run on the QA linux cluster, `alice.sandia.gov`. This script is run on a desktop PC, but is only used to create figures for the analysis report. This script is only included here for completeness.

```python
# this python script is used to plot figures associated
# with SP 9-9: "Analysis Report for Preparation of 2010 Culebra Potentiometric
# Surface Contour Maps."
# by Kris Kuhlman (2011)

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import pyproj

# http://spatialreference.org/ref/epsg/26713/
# http://spatialreference.org/ref/epsg/31013/
putm = pyproj.Proj(init='epsg:26713') # UTM Zone 13N NAD27 (meters)
pstp = pyproj.Proj(init='epsg:32012') # NM state plane east NAD27 (meters)

def transform(xin,yin):
    """does the default conversion from utm -> state plane
    then also convert to feet from meters"""
    xout,yout = pyproj.transform(putm,pstp,xin,yin)
    xout /= M2FT
    yout /= M2FT
    return xout,yout

cfname = 'modeled_head_pest_02.grd'
pfname = 'dtrk_output_pest_02.bln'
wfname = 'modeled_vs_observed_head_pest_02.txt'

M2FT = 0.3048

# read in well-related things
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# load in observed, modeled, obs-mod, (all in meters)
res = np.loadtxt(wfname,skiprows=1,usecols=(3,4,5))
res /= M2FT # convert heads to feet
wellx,welly = transform(*np.loadtxt(wfname,skiprows=1,usecols=(1,2),unpack=True))
names = np.loadtxt(wfname,skiprows=1,usecols=(0,),dtype='|S6')

# read in head-related things
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h = np.loadtxt(cfname,skiprows=5) # ASCII matrix of modeled head in meters AMSL
h[h<0.0] = np.NaN  # no-flow zone in northeast
h[h>1000.0] = np.NaN # constant-head zone in east
h /= M2FT # convert elevations to feet

# surfer grid is implicit in header
# create grid from min/max UTM NAD27 coordinates (meters)
utmy,utmx = np.mgrid[3566500.0:3597100.0:307j, 601700.0:630000.0:284j]

# head contour coords
hx,hy = transform(utmx,utmy)
del utmx,utmy

# read in particle-related things
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
px,py = transform(*np.loadtxt(pfname,skiprows=1,delimiter=',',
                                  usecols=(0,1),unpack=True))
part = np.loadtxt(pfname,skiprows=1,delimiter=',',usecols=(2,))

# read in MODFLOW model, WIPP LWB & ASER contour domain (UTM X & Y)
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
modx,mody =   transform(*np.loadtxt('../../total_boundary.dat',unpack=True))
wippx,wippy = transform(*np.loadtxt('../../wipp_boundary.csv',delimiter=',',
                                                usecols=(1,2),unpack=True))
aserx,asery = transform(*np.loadtxt('../../ASER_boundary.csv',delimiter=',',
                                                usecols=(1,2),unpack=True))

a = []

# plot contour map of entire model area
# *************************************
fig = plt.figure(1,figsize=(12,16))
ax = fig.add_subplot(111)
lev = 3000 + np.arange(17)*10
CS = ax.contour(hx,hy,h,levels=lev,colors='k',linewidths=0.5)
ax.clabel(CS,lev[::2],fmt='%i')
ax.plot(wippx,wippy,'k-')
ax.plot(aserx,asery,'g-')
ax.plot(modx,mody,'-',color='purple',linewidth=2)
ax.plot(wellx,welly,linestyle='none',marker='o',
          markeredgecolor='green',markerfacecolor='none')
ax.set_xticks(630000 + np.arange(10.0)*10000)
ax.set_yticks(450000 + np.arange(10.0)*10000)
labels = ax.get_yticklabels()
for label in labels:
    label.set_rotation(90)
for x,y,n in zip(wellx,welly,names):
    # plot just above
    a.append(plt.annotate(n,xy=(x,y),xytext=(0,5),
                 textcoords='offset points',
                 horizontalalignment='center',
                 fontsize=8))
plt.axis('image')
ax.set_xlabel('NAD27 NM East State Plane Easting (ft)')
ax.set_ylabel('NAD27 NM East State Plane Northing (ft)')

# >>>>>>>manually fix labels>>>>
for lab in a:
    lab.draggable()
plt.show()
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

plt.savefig('large-area-contour-map.png')
plt.close(1)

del lev,CS
mask = np.logical_and(np.logical_and(hx>aserx.min(),hx<aserx.max()),
                      np.logical_and(hy>asery.min(),hy<asery.max()))
h[~mask] = np.NaN

a = []

# plot contour map of ASER-figure area
# *************************************
fig = plt.figure(1,figsize=(12,16))
ax = fig.add_subplot(111)
lev = 3000 + np.arange(17)*5
CS = ax.contour(hx,hy,h,levels=lev,colors='k',linewidths=0.5)
ax.plot(wippx,wippy,'k-')
ax.plot(modx,mody,'-',color='purple',linewidth=2)
ax.plot(wellx,welly,linestyle='none',marker='o',
          markeredgecolor='green',markerfacecolor='none')
ax.plot(px,py,linestyle='solid',color='blue',linewidth=4)
plt.arrow(x=px[-3],y=py[-3],dx=-10,dy=-50,
            linewidth=4,color='blue',head_length=500,head_width=500)
plt.axis('image')
ax.set_xlim([aserx.min(),aserx.max()])
ax.set_ylim([asery.min(),asery.max()])
ax.clabel(CS,lev[::2],fmt='%i',inline_spacing=2)
ax.set_xticks(660000 + np.arange(5.0)*5000)
ax.set_yticks(485000 + np.arange(5.0)*5000)
labels = ax.get_yticklabels()
for label in labels:
    label.set_rotation(90)
for j,(x,y,n) in enumerate(zip(wellx,welly,names)):
    # only plot labels of wells inside the figure area
    if aserx.min()<x<aserx.max() and asery.min()<y<asery.max():
        # name above
        a.append(plt.annotate(n,xy=(x,y),xytext=(0,5),
                    textcoords='offset points',
                    horizontalalignment='center',
                    fontsize=10))
        # observed FW head below
        a.append(plt.annotate('%.1f'%res[j,0],xy=(x,y),xytext=(0,-15),
                    textcoords='offset points',
                    horizontalalignment='center',
                    fontsize=6))
ax.set_xlabel('NAD27 NM East State Plane Easting (ft)')
ax.set_ylabel('NAD27 NM East State Plane Northing (ft)')

# >>>>>>>manually fix labels>>>>
for lab in a:
    lab.draggable()
plt.show()
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

plt.savefig('aser-area-contour-map.png')
plt.close(1)
```