

557633

Analysis Report for Preparation of 2011 Culebra Potentiometric Surface Contour Map

Task Number: 1.4.2.3

Report Date: 5/17/2012

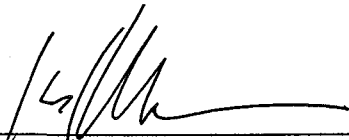


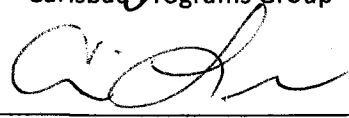
Author:	 _____ Kristopher L. Kuhlman, 6212 Repository Performance Department	<u>5/17/12</u> Date
Technical Review:	 _____ Kevin S. Barnhart, 6212 Repository Performance Department	<u>5/21/12</u> Date
QA Review:	 _____ Shelly R. Nielsen, 6210 Carlsbad Programs Group	<u>5-17-12</u> Date
Management Review:	 _____ Christi D. Leigh, 6212 Manager, Repository Performance Department	<u>5/17/12</u> Date

Table of Contents

1	Introduction.....	3
2	Scientific Approach.....	4
2.1	Modeling Overview	4
2.2	Creating Average MODFLOW Simulation	6
2.3	Boundary Conditions	6
2.4	PEST Calibration of Averaged MODFLOW Model to Observations.....	7
2.5	Figures Generated from Averaged MODFLOW Model	9
3	2011 Results	11
3.1	2011 Freshwater Head Contours	11
3.2	2011 Particle Track	12
3.3	2005 Measured vs. Modeled Fit.....	12
4	References.....	16
5	Run Control Narrative	17
6	Appendix: MODFLOW and Pest Files and Script Source Listings	24
6.1	Input File Listing	24
6.2	Output File Listing	25
6.3	Individual MODFLOW and Pest Script Listings.....	26

1 Introduction

This report documents the preparation of the 2011 potentiometric contour map and associated particle tracks for the Culebra Member of the Rustler Formation in the vicinity of the Waste Isolation Pilot Plant (WIPP), for submittal to the New Mexico Environment Department (NMED). The driver for this analysis is the draft of the Stipulated Final Order sent to NMED on May 28, 2009 (Moody, 2009). This Analysis Report follows the procedure laid out in procedure SP 9-9 (Kuhlman, 2009), which is based upon this NMED driver. This report is similar to Kuhlman (2011); the same analysis is performed on data from 2011, rather than 2010 data. 2011 data for contouring were obtained from the WIPP Management & Operations contractor (Watterson, 2012).

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). For 2011 PEST is used to adjust a subset of the boundary conditions in the averaged MODFLOW model to obtain the best-fit match between the observed freshwater heads and the model-predicted heads. The output of the averaged, PEST-calibrated MODFLOW model is both contoured and used to compute the 2011 advective particle track forward from the WIPP waste-handling shaft.

2 Scientific Approach

2.1 Modeling Overview

Steady-state groundwater flow simulations are carried out using similar software as was used in the analysis report for AP-114 Task 7 (Hart et al., 2009), which was used to create the input calibrated fields. See Table 1 for a summary of all software used in this analysis. The MODFLOW parameter fields (transmissivity (T), anisotropy (A), and recharge (R)) used in this analysis are ensemble averages of the 100 sets of Culebra parameter fields used for WIPP PA for the 2009 Compliance Recertification Application (CRA-2009) PA 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 we derive from the PA MODFLOW model, calibrate using PEST, and use to construct the resulting contour map and particle track, is referred to as the “averaged MODFLOW model.” The PA MODFLOW model T, A and R input fields are appropriately averaged across 100 realizations, producing a single averaged MODFLOW flow model. This averaged MODFLOW model is used to predict regional Culebra groundwater flow across the WIPP site.

For CRA 2009 PABC, PEST was used to construct 100 calibrated model realizations of the PA MODFLOW model by adjusting the spatial distribution of model parameters (T, A, and R); MODFLOW boundary conditions were fixed. The calibration targets for PEST in the PA MODFLOW model were both May 2007 freshwater heads and transient drawdown to large-scale pumping tests. Hart et al. (2009) describe the calibration effort and results that went into the CRA-2009 PABC. An analogous but much simpler process is used here for the averaged MODFLOW model. We use PEST to modify a subset of the MODFLOW boundary conditions (see red boundaries in Figure 1). The boundary conditions are modified, rather than the T, A, and R parameter fields for simplicity, because re-calibrating the 100 T, A, and R parameter fields would be a significant effort (thousands of hours of computer time). The PEST calibration targets for the averaged MODFLOW model are the 2005-2007 measured annual freshwater heads at Culebra monitoring wells. In the averaged MODFLOW model, boundary conditions are modified while holding model parameters T, A, and R constant. In contrast to this, the PA MODFLOW model used fixed boundary conditions and made adjustments to T, A, and R parameter fields.

Table 1. Software used

Software	Version	Description	Platform	Software QA status
MODFLOW-2000	1.6	Flow model	PA cluster	Acquired; qualified under NP 19-1 (Harbaugh et al., 2000)
PEST	9.11	Inverse model	PA cluster	Developed; qualified under NP 19-1 (Doherty, 2002)
DTRKMF	1.00	Particle tracker	PA cluster	Developed; qualified under NP 19-1
Python	2.3.4	Scripting language (file manipulation)	PA cluster	Commercial off the shelf
Enthought Python	7.2-2	Scripting language (plotting)	Mac desktop	Commercial off the shelf
Bash	3.00.15	Scripting language (file manipulation)	PA cluster	Commercial off the shelf

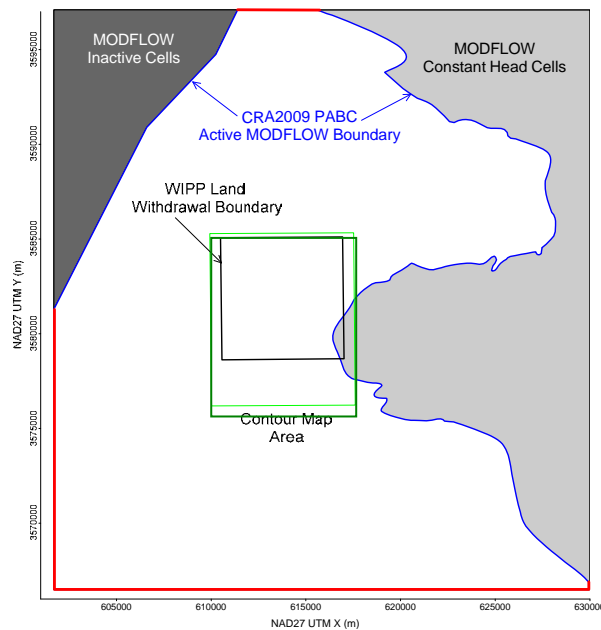


Figure 1. MODFLOW-2000 model domain, adjusted boundary conditions shown in red, contour area outlined in green.

The resulting heads from the PEST-calibrated averaged MODFLOW model are contoured over an area surrounding the WIPP site using matplotlib (a Python plotting library included in the Enthought Python Distribution (EPD)). The figure covers a subset of the complete MODFLOW model domain – see the green rectangle surrounding the WIPP LWB in Figure 1. We compute the path taken by a conservative (i.e., non-dispersive and non-reactive) particle to the WIPP LWB, initially released to the Culebra at the waste-handling shaft. The particle track is computed from the MODFLOW flow field using DTRKMF, these results are also plotted using matplotlib. Scatter plot statistics were computed using NumPy (an array-functionality Python library included in EPD), which summarize the quality of the fit between the

averaged MODFLOW model and observed Culebra freshwater heads. 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 scripts on an Intel-Xeon-equipped desktop computer running Mac OS X, version 10.6.8.

2.2 Creating Average MODFLOW Simulation

An averaged MODFLOW model is used to compute the freshwater head and cell-by-cell flow vectors. The heads are contoured and the flow vectors are used to compute particle tracks. The ensemble-averaged inputs are used to create a single average simulation that produces a single averaged output, rather than averaging the 100 individual outputs of the Culebra flow model used for WIPP PA. This approach was taken to simplify the contouring process, and create a single contour map that exhibits physically realistic patterns (i.e., its behavior is constrained by the groundwater flow equation). The alternative approach would be to average outputs from 100 models to produce a single average result, but the result may be physically unrealistic. The choice to average inputs, rather than outputs, is a simplification (only one model must be calibrated using PEST, rather than 100) that results in smoother freshwater head contours and faster particle tracks, compared to those predicted by the ensemble of fields in AP114 Task 7 (Hart et al., 2009).

The MODFLOW model grid is a single layer, comprised of 307 rows and 284 columns, each model cell being a 100-meter 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 Universal Transverse Mercator (UTM) North American Datum 1927 (NAD27) coordinates, zone 13.

The calibrated T, A, and R parameter fields from the PA MODFLOW model were checked out of the PA repository using the `checkout_average_run_modflow.sh` script (scripts are listed completely in the Appendix; input and output files are available from the WIPP version control system in the repository `$CVSLIB/Analyses/SP9_9`). Model inputs can be divided into two groups. The first group includes 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 includes the T, A, and R fields, which are different for each realization. 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 arithmetically averaged in \log_{10} space, 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 (the boundary along the dark gray region in the upper left of Figure 1). The constant-head boundary along the halite margin corresponds to a geologic boundary (the eastern

irregular boundary adjoining the light gray region in the right of 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. This type of boundary includes the linear southern, southwestern, and northern boundaries that coincide with the rectangular frame surrounding the model domain (shown as heavy red lines 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 meters above mean sea level (AMSL) are left fixed and not adjusted in the PEST optimization, because they correspond to the land surface. 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 input type includes the observed freshwater head values, which are used as targets for the PEST calibration. The second input class 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. These files comprise the forward model that PEST runs repeatedly to estimate sensitivities of model outputs to model inputs. The third input type includes the PEST configuration files, which list parameter and observation groups, observation weights, and indicate which parameters in the MODFLOW model will be adjusted in the inverse simulation. Freshwater head values used as targets for the PEST calibration from Watterson (2012) are listed in Table 2.

Table 2. Freshwater Head Calibration Targets used in PEST, from Waterson (2012).

Well	Measurement Date	Freshwater Head Elevation (m AMSL)
AEC-7	08/09/11	934.139
C-2737	08/10/11	921.203
ERDA-9	08/10/11	925.078
H-02b2	08/10/11	928.603
H-03b2	08/10/11	918.983
H-04bR	08/09/11	916.623
H-05b	08/09/11	939.440
H-06bR	08/09/11	936.075
H-07b1	08/08/11	913.907
H-09bR	08/08/11	912.934
H-10c	08/09/11	922.942
H-11b4	08/09/11	915.444
H-12	08/09/11	917.918
H-15R	08/10/11	920.024
H-16	08/10/11	929.407
H-17	08/09/11	916.959
H-19b0	08/10/11	919.032
IMC-461	08/08/11	927.640
SNL-01	08/08/11	939.963
SNL-02	08/08/11	936.192
SNL-03	08/09/11	939.233
SNL-05	08/08/11	937.088
SNL-08	08/09/11	930.440
SNL-09	08/09/11	930.853
SNL-10	08/08/11	930.827
SNL-12	08/08/11	915.185
SNL-14	08/09/11	915.822
SNL-16	08/08/11	916.972
SNL-17	08/08/11	916.127
SNL-18	08/08/11	937.179
SNL-19	08/08/11	936.213
WIPP-11	08/09/11	939.535
WIPP-13	08/10/11	938.283
WIPP-19	08/10/11	934.081
WQSP-1	08/10/11	937.824
WQSP-2	08/10/11	940.059
WQSP-3	08/10/11	937.272
WQSP-4	08/10/11	919.488
WQSP-5	08/10/11	918.690

Information Only

WQSP-6	08/10/11	922.167
--------	----------	---------

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:

$$h(x, y) = A + B(y + D \text{sign}(y) \text{abs}(y)^\alpha) + C(Ex^3 + Fx^2 - x) \quad (1)$$

where $\text{abs}(y)$ is absolute value and $\text{sign}(y)$ 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 $-1 \leq \{x, y\} \leq 1$. In Hart et al. (2009), the values $A=928.0$, $B=8.0$, $C=1.2$, $D=1.0$, $\alpha=0.5$, $E=1.0$, and $F=-1.0$ are used with the above equation to assign the boundary conditions.

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 α . Each forward run of the model corresponded to a call to the Bash script `run_02_model`. This script called the `surface_02_extrapolate.py` script, the MODFLOW-2000 executable, and the 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), and the PEST control file (listing the ranges and initial values for the estimable parameters and the weights associated with observations). The wells used in each year's PEST calibration were separated into three groups. Higher weights (2.5) were assigned to wells inside the LWB, and lower weights (0.4) were assigned to wells distant to the WIPP site, while wells in the middle were assigned an intermediate weight (1.0). 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. The additional observations and weights were assigned to improve the fit in the area of interest (inside the WIPP LWB), possibly at the expense of a somewhat poorer fit closer to the boundary conditions.

2.5 Figures Generated from Averaged MODFLOW Model

The MODFLOW model is run predictively using the averaged MODFLOW 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 the WIPP LWB. Particle tracking stops when the particle crosses the WIPP LWB. The Python script `convert_dtrkmf_output_for_surfer.py` converts the MODFLOW cell-indexed 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 matrix file format using the Python script `head_bin2ascii.py`.

The resulting particle track and contours of the model-predicted head are plotted using a matplotlib Python script for an area including the WIPP LWB, corresponding to the region shown in previous versions of the ASER (e.g., see Figure 6.11 in DOE (2008)), specifically the green box 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`.

Information Only

3 2011 Results

3.1 2011 Freshwater Head Contours

The model-generated freshwater head contours are given in Figure 2 and Figure 3. There is a roughly east-west trending band of steeper gradients, corresponding to lower Culebra transmissivity. The uncontoured region in the eastern part of the figures 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” has a high freshwater head but extremely low transmissivity, essentially serving as a no-flow boundary in this area.

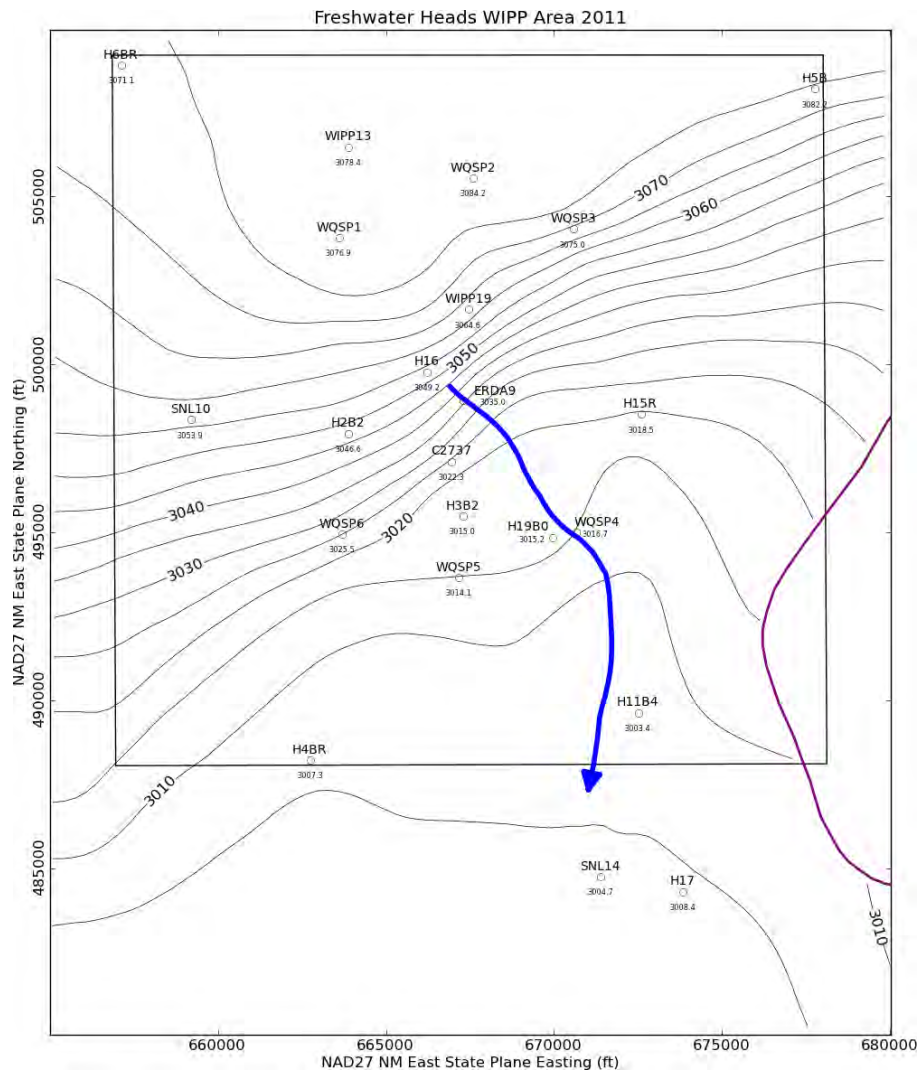


Figure 2. Model-generated August 2011 freshwater head contours with observed head listed at each well (5-foot contour interval) with blue water particle track from waste handling shaft to WIPP LWB. Purple curve is Rustler halite margin.

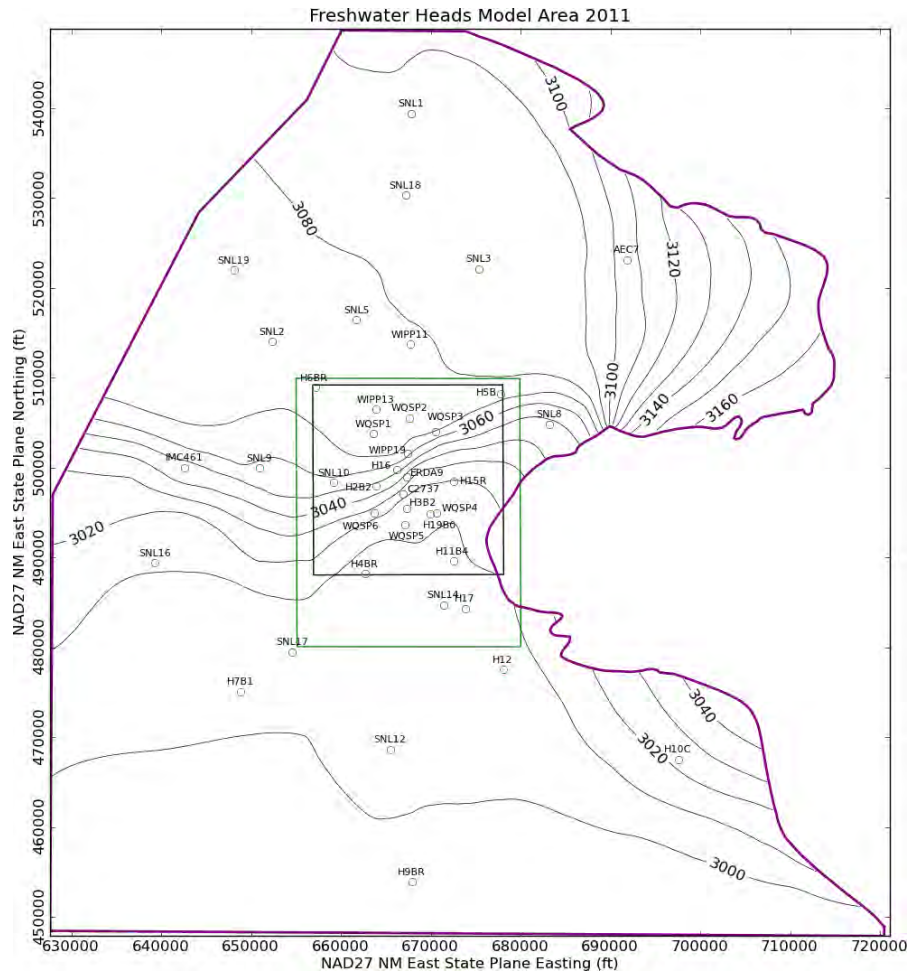


Figure 3. MODFLOW-modeled August 2011 heads for entire model domain (10-foot contour interval). Green rectangle indicates region contoured in Figure 2, black square is WIPP LWB.

3.2 2011 Particle Track

The blue arrow in Figure 2 shows the DTRKMF-calculated path a water particle would take through the Culebra from the coordinates corresponding to the WIPP waste handling shaft to the LWB (a path length of 4092 m). Assuming a 4-m thickness for the transmissive portion of the Culebra and a constant porosity of 16%, the travel time to the WIPP LWB is 5826 years (output from DTRKMF is adjusted from an original 7.75-m Culebra thickness). This is an average velocity of 0.70 m/yr.

3.3 2011 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 'x's, and other wells within the MODFLOW model domain but distant from the WIPP site are given by a blue star. AEC-7 was given a low weight (0.01), to prevent its large residual from dominating the optimization. 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).

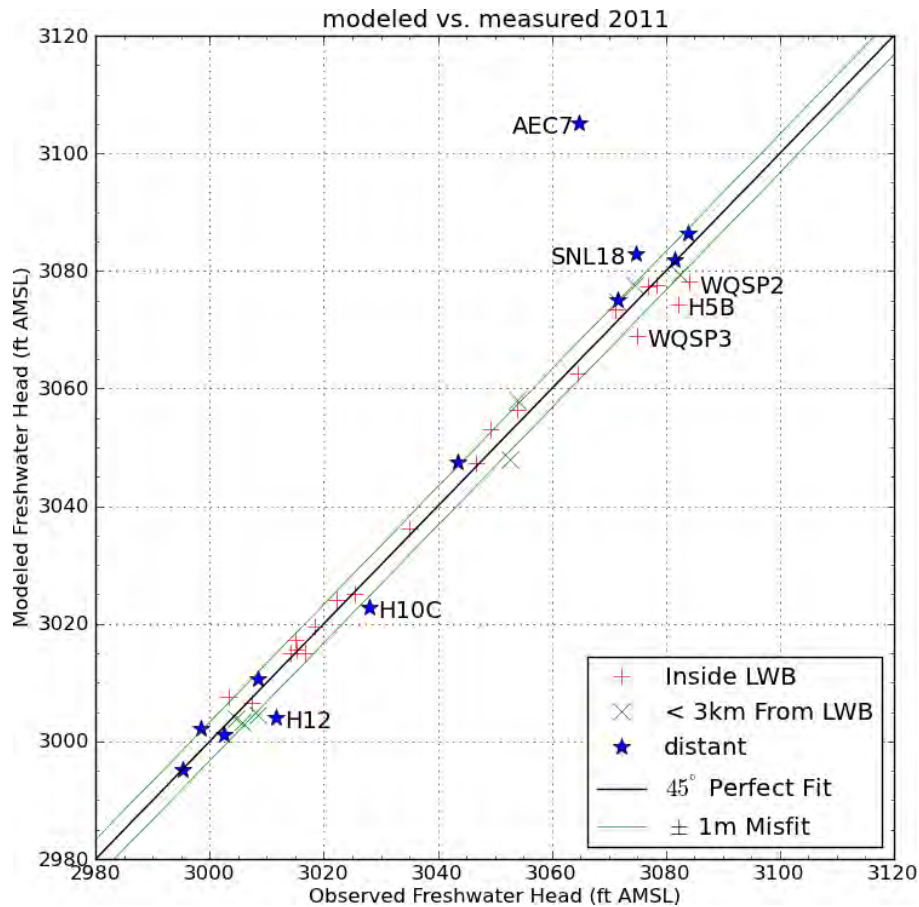


Figure 4. Measured vs. modeled scatter plot for averaged MODFLOW model generated heads and August 2011 observed freshwater heads

The central black diagonal line in Figure 4 represents a perfect model fit (1:1 or 45-degree slope); the two green 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 is not included on Figure 4). The calibrated parameters (for equation 1) were A=928.8, B=7.97, C=1.21, D=0.943, E=1.06, F=-1.05, and $\alpha=0.227$. The parameter α , the exponent associated with the y-direction term, had the largest relative change (-55%) compared to the starting values. All other parameters were within 6% of their original values.

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 model fit to observed data. The wells within and near the WIPP LWB have an R^2 of greater than 94%, and the calibration improved

the R² value very slightly (third decimal place) inside the WIPP LWB. The distribution in Figure 5 does not have a strong bias.

Table 3. 2011 Measured vs. Modeled correlation coefficients

	dataset	measured vs. modeled R ²
Uncalibrated	wells inside WIPP LWB	0.989
	wells <3km from WIPP LWB	0.987
	all wells	0.949
Calibrated	wells inside WIPP LWB	0.990
	wells <3km from WIPP LWB	0.987
	all wells	0.949

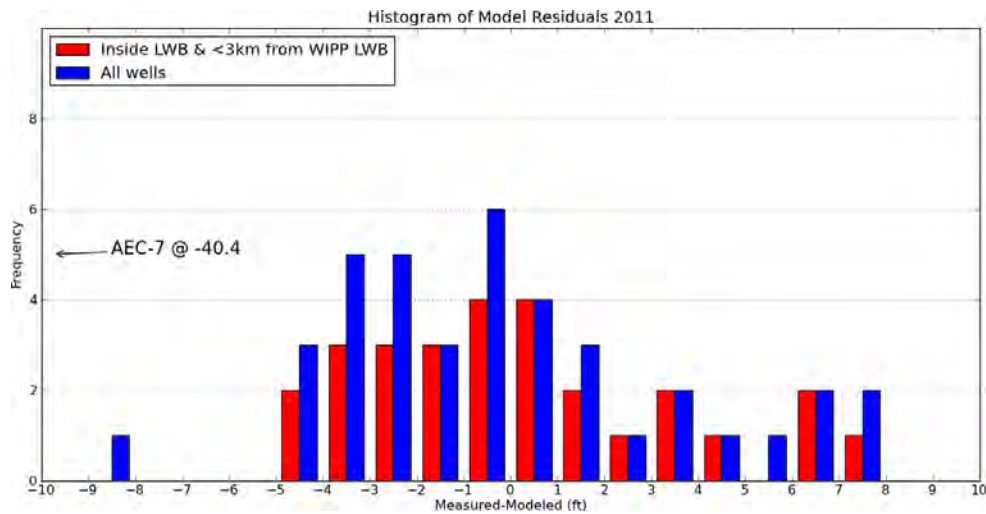


Figure 5. Histogram of Measured-Modeled errors for 2011

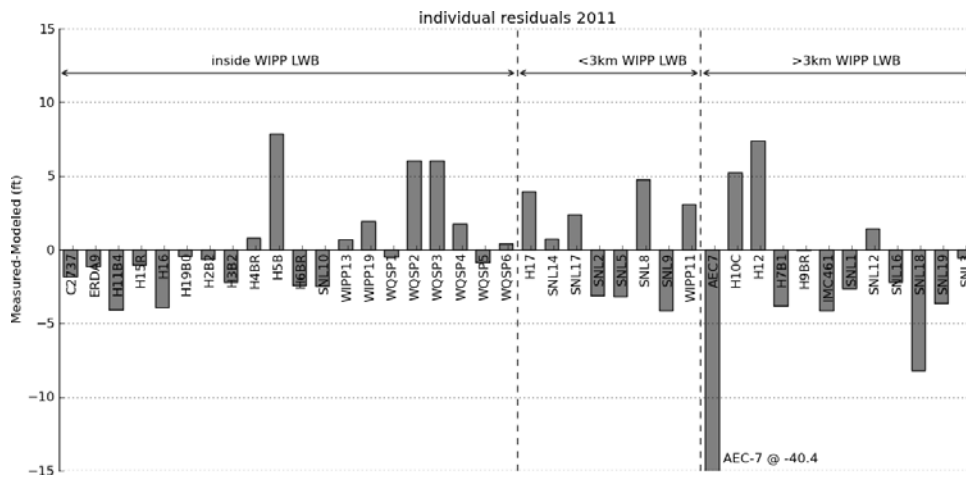


Figure 6. Measured-Modeled errors at each well location for 2011

Aside from AEC-7, the model fit to the August 2011 observations is very good. The averaged MODFLOW model captures the bulk Culebra flow behavior, while the PEST calibration improved the model fit to the specific August 2011 observations.

Information Only

4 References

- 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.
- Johnson, P.B. 2008. *Potentiometric Surface, Adjusted to Equivalent Freshwater Heads, of the Culebra Dolomite Member of the Rustler Formation near the WIPP Site, May 2007 (AP-114 Task 6)*. Carlsbad, NM, Sandia National Laboratories, ERMS 548746.
- Johnson, P.B. 2009. *Potentiometric Surface, Adjusted to Equivalent Freshwater Heads, of the Culebra Dolomite Member of the Rustler Formation near the WIPP Site, May 2007, Revision 2 (AP-114 Task 6)*. Carlsbad, NM, Sandia National Laboratories, ERMS 551116.
- Kuhlman, K.L. 2011. *Analysis Report for Preparation of 2010 Culebra Potentiometric Surface Contour Map, Rev 2*, Sandia National Laboratories, Carlsbad, NM, ERMS 555318.
- 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. 2012. *August 2011 Culebra ASER map data*, Washington TRU Solutions, Carlsbad, NM. WIPP Records Center, ERMS 557462.

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` (SA-4.1); this script first exports the 3 parameter fields (transmissivity (T), anisotropy (A), and 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` (SA-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.

The input files used by this analysis, the output files from this analysis (including the plotting scripts) are checked into the WIPP version control system (CVS) under the repository `$CVSLIB/Analyses/SP9_9`.

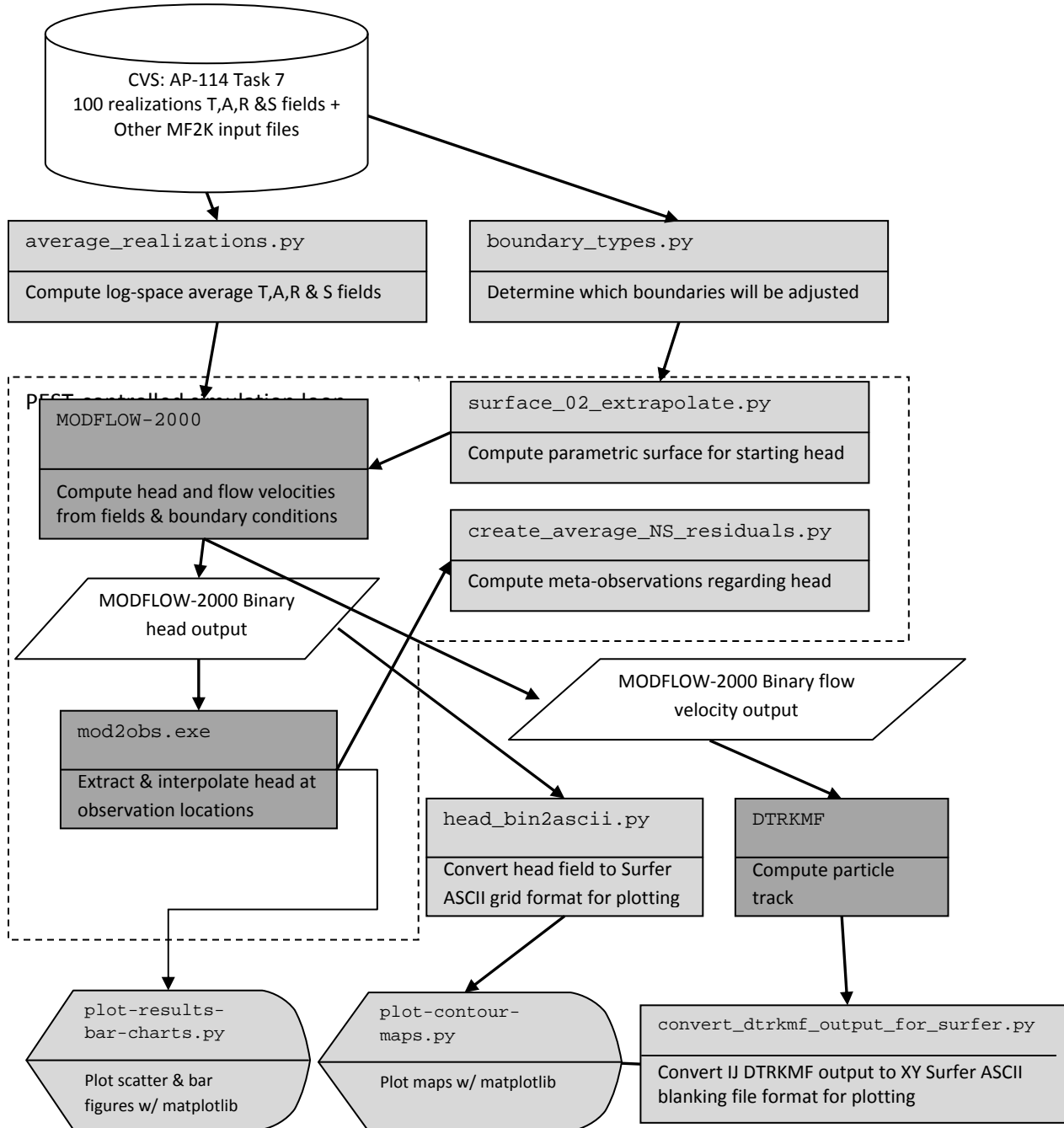


Figure 7. Process flowchart; dark gray indicates qualified programs, light gray are scripts written for this analysis

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.

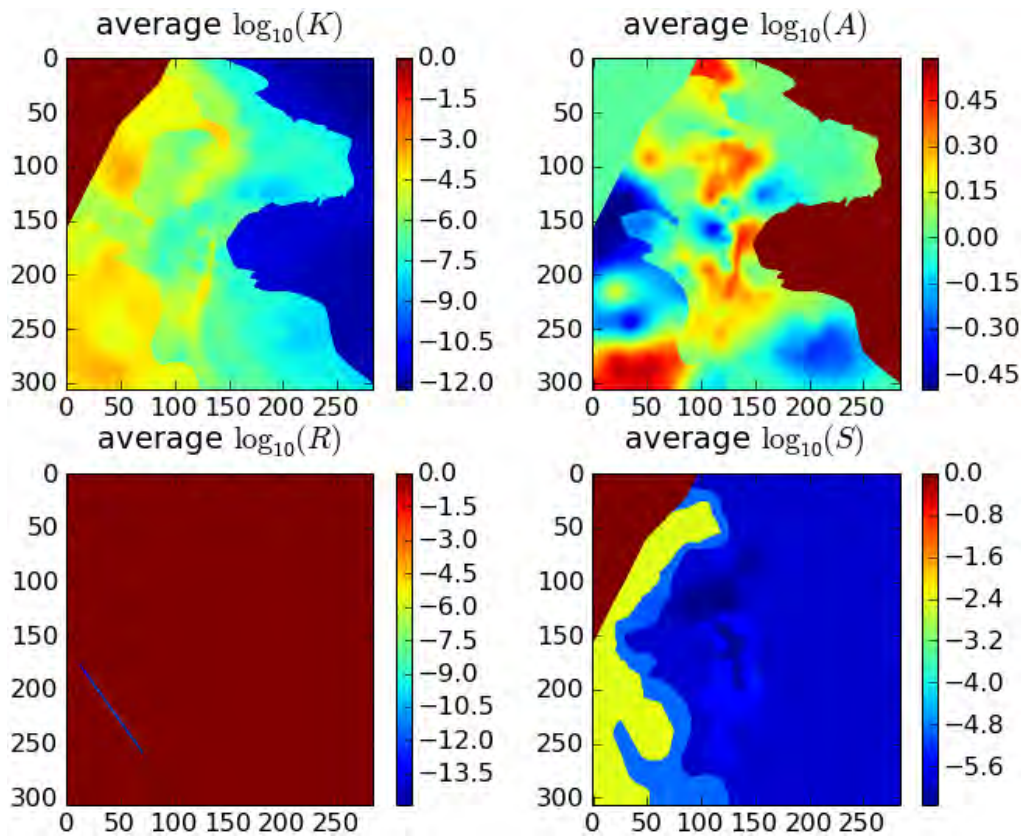


Figure 8. Plots of base-10 logarithms of average parameter fields; rows and columns are labeled on edges of figures.

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 Python using matplotlib, 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.blm`. 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 than the original cumulative travel time by the factor $7.75/4=1.9375$). The conversion from row, column to X, Y is

$$X = 601700.0 + 100.0 * \text{column}$$

$$Y = 3597100.0 - 100.0 * \text{row}$$

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 Python using matplotlib, 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							
0.0000000E+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	0.0000000E+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, 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.

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_2011ASER.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_2011ASER.smp`), and the weights are read in from the input file (`obs_loc_2011ASER.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.

Information Only

6 Files and Script Source Listings

6.1 Input Files

bytes	file type	description	file name
1.4K	Python script	average 100 realizations	average_realizations.py
2.0K	Python script	distinguish different BC types	boundary_types.py
6.5K	Bash script	main routine: checkout files, run MODFLOW run PEST, call Python scripts	checkout_average_run_modflow.sh
624	Python script	convert DTRKMF IJ output to Surfer X,Y blanking format	convert_dtrkmf_output_for_surfer.py
2.7K	Python script	create meta observations of avg heat	create_average_NS_residuals.py
3.0K	Python script	create PEST input files from observed data	create_pest_02_input.py
48	input listing	responses to DTRKMF prompts	dtrkmf.in
3.9K	Python script	convert MODFLOW binary output to Surfer ASCII grid format	head_bin2ascii.py
1.1K	input	listing of 100 realizations from CVS	keepers
1.5K	input	observed August 2011 heads in mod2obs.exe bore sample file format	meas_head_2011ASER.smp
968	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
139	input listing	responses to mod2obs.exe prompts	mod2obs_head.in
372	file listing	files needed to run MODFLOW	modflow_files.dat
382	input	listing of wells and geographic groupings	obs_loc_2011ASER.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
2.4K	Python script	compute starting head from parameter and coordinate inputs	surface_02_extrapolate.py
506	input	DTRKMF input file	wippctrl.inp

Table 1: Input Files

6.2 Output Files

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.blm
16K	PEST output	matrix condition numbers	bc_adjust_2011ASER.cnd
2.7K	PEST output	binary intermediate file	bc_adjust_2011ASER.drf
7.4K	PEST output	binary intermediate file	bc_adjust_2011ASER.jac
7.5K	PEST output	binary intermediate file	bc_adjust_2011ASER.jco
9.9K	PEST output	binary intermediate file	bc_adjust_2011ASER.jst
3.8K	PEST output	parameter statistical matrices	bc_adjust_2011ASER.mtt
477	PEST output	parameter file	bc_adjust_2011ASER.par
62K	PEST output	optimization record	bc_adjust_2011ASER.rec
4.6K	PEST output	model outputs for last iteration	bc_adjust_2011ASER.rei
8.4K	PEST output	summary of residuals	bc_adjust_2011ASER.res
28	PEST output	binary restart file	bc_adjust_2011ASER.rst
24K	PEST output	relative parameter sensitivities	bc_adjust_2011ASER.sen
4.0K	PEST output	absolute parameter sensitivities	bc_adjust_2011ASER.seo
213K	png image	matplotlib plot (Fig. 2)	aser-area-contour-map2011.png
223K	png image	matplotlib plot (Fig. 3)	large-area-contour-map2011.png
33K	png image	matplotlib plot (Fig. 5)	model-error-histogram2011.png
55K	png image	matplotlib plot (Fig. 6)	model-error-residuals2011.png
93K	png image	matplotlib plot (Fig. 4)	scatter_pest_02_2011.png

Table 2: Listing of Output Files

6.3 Individual scripts

6.3.1 Bash shell script checkout_average_run_modflow.sh

```
1 #!/bin/bash
2
3 set -o nounset # explode if using an un-initialized variable
4 set -o errexit # exit on non-zero error status of sub-command
5
6 # this script makes the following directory substructure
7 #
8 # current_dir \----- Outputs (calibrated parameter fields - INPUTS)
9 #              \----- Inputs (other modflow files - INPUTS)
10 #              \----- original_average (foward sim using average fields)
11 #              \----- bin (MODFLOW and DTRKMF binaries)
12 #              \----- pest_0? (pest-adjusted results)
13
14 ##set -o xtrace
15
16 echo " ~~~~~ "
17 echo " checking out T fields"
18 echo " ~~~~~ "
19
20 # these will checkout the calibrated parameter-field data into subdirectories
21 # checkout things that are different for each of the 100 realizaions
22 for d in `cat keepers`
23 do
24     cvs -d /nfs/data/CVSLIB/Tfields checkout Outputs/${d}/modeled_{K,A,R,S}_field.mod
25 done
26
27 # checkout MODFLOW input files that are constant for across all realizations
28 cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/data/elev_{top,bot}.mod
29 cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/data/init_{bnds.inf,head.mod}
30 cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/modflow/mf2k_culebra.{lmg,lpf}
31 cvs -d /nfs/data/CVSLIB/Tfields checkout Inputs/modflow/mf2k_head.{ba6,nam,oc,dis,rch}
32
33 # modify the path of "updated" T-fields, so they are all at the
34 # same level in the directory structure (simplifying scripts elsewhere)
35
36 if [ -a keepers_short ]
37 then
38     rm keepers_short
39 fi
40 touch keepers_short
41
42 for d in `cat keepers`
43 do
44     bn=`basename ${d}`
45     # test whether it is a compount path
46     if [ ${d} != ${bn} ]
47     then
48         dn=`dirname ${d}`
```

```

49     mv ./Outputs/${d} ./Outputs/
50
51     # put an empty file in the directory to indicate
52     # what the directory was previously named
53     touch ./Outputs/${bn}/${dn}
54 fi
55
56 # create a keepers list without directories
57 echo ${bn} >> keepers_short
58 done
59
60 # -----
61
62 echo " ~~~~~ "
63 echo " perform averaging across all realizations "
64 echo " ~~~~~ "
65
66 python average_realizations.py
67
68 # checkout MODFLOW / DTRKMF executables
69 cvs -d /nfs/data/CVSLIB/MODFLOW2K checkout bin/mf2k/mf2k_1.6.release
70 cvs -d /nfs/data/CVSLIB/MODFLOW2K checkout bin/dtrkmf/dtrkmf_v0100
71
72 # check out pest and obs2mod binaries
73 cd bin
74 cvs -d /nfs/data/CVSLIB/PEST checkout Builds/Linux/pest.exe
75 cvs -d /nfs/data/CVSLIB/PEST checkout Builds/Linux/mod2obs.exe
76 cd ..
77
78 # -----
79
80 echo " ~~~~~ "
81 echo " setup copies of files constant between all realizations "
82 echo " ~~~~~ "
83
84 # directory for putting original base-case results in
85 od=original_average
86
87 if [ -d ${od} ]
88 then
89     echo ${od}" directory exists: removing and re-creating"
90     rm -rf ${od}
91 fi
92
93 mkdir ${od}
94 cd ${od}
95 echo `pwd`
96
97 # link to unchanged input files
98 for file in `cat ../modflow_files.dat`
99 do

```

```

100     ln -sf ${file} .
101 done
102
103 # link to averaged files computed in previous step
104 for f in {A,R,K,S}
105 do
106     ln -sf ../modeled-${f}_field.avg ../modeled-${f}_field.mod
107 done
108
109 ln -sf elev_top.mod fort.33
110 ln -sf elev_bot.mod fort.34
111
112 echo " ~~~~~~"
113 echo " run original MODFLOW and DTRKMF and export results for plotting"
114 echo " ~~~~~~"
115
116 # run MODFLOW, producing average head and CCF
117 ../bin/mf2k/mf2k-1.6.release mf2k_head.nam
118
119 # run DTRKMF, producing particle track (from ccf)
120 ../bin/dtrkmf/dtrkmf_v0100 <dtrkmf.in
121
122 # convert binary MODFLOW head output to Surfer ascii grid file format
123 ln -sf ../head_bin2ascii.py .
124 python head_bin2ascii.py
125 mv modeled_head_asiihed.grd modeled_head-${od}.grd
126
127 # convert DTRKMF output from cells to X,Y and
128 # save in Surfer blanking file format
129 ln -sf ../convert_dtrkmf_output_for_surfer.py .
130 python convert_dtrkmf_output_for_surfer.py
131 mv dtrk_output.blm dtrk_output-${od}.blm
132
133 # extract head results at well locations and merge with observed
134 # head file for easy scatter plotting in Excel (tab delimited)
135 for file in `cat ../mod2obs_files.dat`
136 do
137     ln -sf ${file} .
138 done
139
140 ln -sf ../meas_head_2011ASER.smp .
141 ln -sf ../obs_loc_2011ASER.dat .
142 ../bin/Builds/Linux/mod2obs.exe <mod2obs_head.in
143 ln -sf ../merge_observed_modeled_heads.py
144 python merge_observed_modeled_heads.py
145 mv both_heads.smp modeled_vs_observed_head-${od}.txt
146
147
148 # go back down into root directory
149 cd ..
150 echo `pwd`

```

```

151
152 echo " ~~~~~~"
153 echo " setup and run PEST to optimize parametric surface to set BC "
154 echo " ~~~~~~"
155
156 for p in pest_02
157 do
158
159     if [ -d ${p} ]
160     then
161         echo ${p}" directory exists: removing and re-creating"
162         rm -rf ${p}
163     fi
164
165     mkdir ${p}
166     cd ${p}
167     echo `pwd`
168
169     # link to unchanged input files
170     for file in `cat ../modflow_files.dat`
171     do
172         ln -sf ${file} .
173     done
174
175     # link to averaged files computed in previous step
176     for f in {A,R,K,S}
177     do
178         ln -sf ../modeled_${f}_field.avg ./modeled_${f}_field.mod
179     done
180
181     # link to mod2obs files (needed for pest)
182     for file in `cat ../mod2obs_files.dat`
183     do
184         ln -sf ${file} .
185     done
186
187     # link to pest files
188     for file in `cat ../${p}_files.dat`
189     do
190         ln -s ${file} .
191     done
192
193     # rename 'original' versions of files to be modified by pest
194     rm init_head.mod
195     ln -sf ../Inputs/data/init_head.mod ./init_head_orig.mod
196     rm init_bnds.inf
197     ln -sf ../Inputs/data/init_bnds.inf ./init_bnds_orig.inf
198
199     # create new ibound array for easier modification during PEST
200     # optimization iterations
201     python boundary_types.py

```

```

202
203 # create the necessary input files from observations
204 python create_${p}_input.py
205
206 # run pest
207 ../bin/Builds/Linux/pest.exe bc_adjust_2011ASER
208
209 # last output files should be best run
210 # extract all the stuff from that output
211 #####
212
213 ln -sf elev_top.mod fort.33
214 ln -sf elev_bot.mod fort.34
215
216 ../bin/dtrkmf/dtrkmf_v0100 <dtrkmf.in
217
218 ln -sf ../head_bin2ascii.py .
219 python head_bin2ascii.py
220 mv modeled_head_asiihed.grd modeled_head_${p}.grd
221
222 ln -sf ../convert_dtrkmf_output_for_surfer.py .
223 python convert_dtrkmf_output_for_surfer.py
224 mv dtrk_output.blm dtrk_output_${p}.blm
225
226 for file in `cat ../mod2obs_files.dat`
227 do
228     ln -sf ${file} .
229 done
230
231 ../bin/Builds/Linux/mod2obs.exe <mod2obs_head.in
232 ln -sf ../merge_observed_modeled_heads.py
233 python merge_observed_modeled_heads.py
234 mv both_heads.smp modeled_vs_observed_head_${p}.txt
235
236 cd ..
237 done

```

6.3.2 Python script average_realizations.py

```
1 from math import log10, pow
2
3 nrow = 307
4 ncol = 284
5 nel = nrow*ncol
6 nfr = 100 # number of fields (realizations)
7 nft = 4   # number of field types
8
9 def floatload(filename):
10     """Reads file (a list of strings, one per row) into a list of strings."""
11     f = open(filename, 'r')
12     m = [float(line.rstrip()) for line in f]
13     f.close()
14     return m
15
16 types = ['K', 'A', 'R', 'S']
17
18 # get list of 100 best calibrated fields
19 flist = open('keepers_short', 'r')
20 runs = flist.read().strip().split('\n')
21 flist.close()
22
23 # initialize to help speed lists up a bit
24 # nfr (100) realizations of each
25 fields = []
26 for i in xrange(nft):
27     fields.append([None]*nfr)
28     for i in xrange(nfr):
29         # each realization being nel (87188) elements
30         fields[-1][i] = [None]*nel
31
32 # read in all realizations
33 print 'reading ...'
34 for i, run in enumerate(runs):
35     print i, run
36     for j, t in enumerate(types):
37         fields[j][i][0:nel] = floatload('Outputs/'+ run + '/modeled_' + t + '_field.mod')
38
39 # open up files for writing
40 fh = []
41 for t in types:
42     fh.append(open('modeled_' + t + '_field.avg', 'w'))
43
44 # transpose fields to allow slicing across realizations, rather than across cells
45 for j in range(len(types)):
46     fields[j] = zip(*(fields[j]))
47
48 print 'writing ...'
49 # do averaging across 100 realizations
```

```
50 for i in xrange(nel):
51     if i%10000 == 0:
52         print i
53     for h,d in zip(fh, fields):
54         h.write('%18.11e\n' % pow(10.0, sum(map(log10, d[i])))/ nfr )
55
56 for h in fh:
57     h.close()
```


6.3.3 Python script boundary_types.py

```
1 nx = 284      # number columns in model grid
2 ny = 307      # number rows
3 nel = nx*ny
4
5 def intload(filename):
6     """Reads file (a 2D integer array) as a list of lists.
7     Outer list is rows, inner lists are columns."""
8     f = open(filename, 'r')
9     m = [[int(v) for v in line.rstrip().split()] for line in f]
10    f.close()
11    return m
12
13 def intsave(filename, m):
14    """Writes file as a list of lists as a 2D integer array, format '%3i'.
15    Outer list is rows, inner lists are columns."""
16    f = open(filename, 'w')
17    for row in m:
18        f.write(' '.join(['%2i' % col for col in row]) + '\n')
19    f.close()
20
21 def floatload(filename):
22    """Reads file (a list of real numbers, one number each row) into a list of floats."""
23    f = open(filename, 'r')
24    m = [float(line.rstrip()) for line in f]
25    f.close()
26    return m
27
28 def reshapev2m(v):
29    """Reshape a vector that was previously reshaped in C-major order from a matrix,
30    back into a matrix (here a list of lists)."""
31    m = [None]*ny
32    for i, (lo, hi) in enumerate(zip(xrange(0, nel-nx+1, nx), xrange(nx, nel+1, nx))):
33        m[i] = v[lo:hi]
34    return m
35
36 #####
37
38 # read in original MODFLOW IBOUND array (only 0,1, and -1)
39 ibound = intload('init_bnds_orig.inf')
40
41 # read in initial heads
42 h = reshapev2m(floatload('init_head_orig.mod'))
43
44 # discriminate between two types of constant head boundaries
45 # -1) CH, where value > 1000 (area east of halite margin)
46 # -2) CH, where value < 1000 (single row/column of cells along edge of domain)
47
48 for i, row in enumerate(ibound):
49     for j, val in enumerate(row):
```

50
51
52
53
54
55
56
57

```
# 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 dur  
# 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

```
1 prefix = '2011ASER'
2
3 #####
4 ## pest instruction file reads output from mod2obs
5 fin = open('meas_head_%s.smp' % prefix, 'r')
6
7 # each well is a [name,head] pair
8 wells = [[line.split()[0],line.split()[3]] for line in fin]
9 fin.close()
10
11 fout = open('modeled_head.ins','w')
12 fout.write('pif @\n')
13 for i,well in enumerate(wells):
14     fout.write("l1 [%s]39:46\n" % well[0])
15 fout.close()
16
17 # exponential surface used to set initial head everywhere
18 # except east of the halite margins, where the land surface is used.
19 # initial guesses come from AP-114 Task report
20 params = [928.0, 8.0, 1.2, 1.0, 1.0, -1.0, 0.5]
21 pnames = ['a', 'b', 'c', 'd', 'e', 'f', 'exp']
22
23 fout = open('avg_NS_res.ins','w')
24 fout.write("""pif @
25 l1 [medianN]1:16
26 l1 [medianS]1:16
27 l1 [meanN]1:16
28 l1 [meanS]1:16
29 """)
30 fout.close()
31
32
33 #####
34 ## pest template file
35 ftmp = open('surface_par_params.ptf','w')
36 ftmp.write('ptf @\n')
37 for n in pnames:
38     ftmp.write('@ %s @\n' % n)
39 ftmp.close()
40
41
42 #####
43 ## pest parameter file
44
45 fpar = open('surface_par_params.par','w')
46 fpar.write('double point\n')
47 for n,p in zip(pnames,params):
48     fpar.write('%s %.2f 1.0 0.0\n' % (n,p))
49 fpar.close()
```

```

50
51
52 #####
53 ## pest control file
54
55 f = open('bc_adjust_%s.pst' % prefix, 'w')
56
57 f.write("""pcf
58 * control data
59 restart estimation
60 %i %i 1 0 2
61 1 2 double point 1 0 0
62 5.0 2.0 0.4 0.001 10
63 3.0 3.0 1.0E-3
64 0.1
65 30 0.001 4 4 0.0001 4
66 1 1 1
67 * parameter groups
68 bc relative 0.005 0.0001 switch 2.0 parabolic
69 """ % (len(params), len(wells)+4))
70
71 f.write('* parameter data\n')
72 for n,p in zip(pnames, params):
73     if p > 0:
74         f.write('%s none relative %.3f %.3f %.3f bc 1.0 0.0 1\n' %
75                (n, p, -2.0*p, 3.0*p))
76     else:
77         f.write('%s none relative %.3f %.3f %.3f bc 1.0 0.0 1\n' %
78                (n, p, 3.0*p, -2.0*p))
79
80 f.write("""* observation groups
81 ss_head
82 avg_head
83 * observation data
84 """)
85
86 ## read in observation weighting group definitions
87 fin = open('obs_loc_%s.dat' % prefix, 'r')
88 location = [line.rstrip().split()[1] for line in fin]
89 fin.close()
90
91 weights = []
92
93 for l in location:
94     # inside LWB
95     if l == '0':
96         weights.append(2.5)
97     # near LWB
98     if l == '1':
99         weights.append(1.0)
100    # distant to LWB

```

```

101     if l == '2':
102         weights.append(0.4)
103     if l == '99':
104         weights.append(0.01) # AEC-7
105
106
107 for name, head, w in zip(zip(*wells)[0], zip(*wells)[1], weights):
108     f.write('%s %s %.3f ss_head\n' % (name, head, w))
109
110 # one fewer N observation (WIPP-25 removed), there were 13
111 # there are 12 N observations in the average and 11 S, therefore
112 # split the weight between the mean and median
113 f.write("""medianN 0.0 18.0 avg_head
114 medianS 0.0 16.5 avg_head
115 meanN 0.0 18.0 avg_head
116 meanS 0.0 16.5 avg_head
117 """)
118
119 f.write("""* model command line
120 ./run_02_model
121 * model input/output
122 surface_par_params.ptf surface_par_params.in
123 modeled_head.ins modeled_head.smp
124 avg_NS_res.ins avg_NS_res.smp
125 """)
126 f.close()

```

6.3.5 Python script surface_02_extrapolate.py

```
1 from itertools import chain
2 from math import sqrt
3
4 def matload(filename):
5     """Reads file (a 2D string array) as a list of lists.
6     Outer list is rows, inner lists are columns."""
7     f = open(filename, 'r')
8     m = [line.rstrip().split() for line in f]
9     f.close()
10    return m
11
12 def floatload(filename):
13     """Reads file (a list of real numbers, one number each row) into a list of floats."""
14     f = open(filename, 'r')
15     m = [float(line.rstrip()) for line in f]
16     f.close()
17    return m
18
19 def reshapem2v(m):
20     """Reshapes a rectangular matrix into a vector in same fashion as numpy.reshape().
21     which is C-major order"""
22    return list(chain(*m))
23
24 def sign(x):
25     """ sign function """
26     if x<0:
27         return -1
28     elif x>0:
29         return +1
30     else:
31         return 0
32
33 #####
34
35 # read in modified IBOUND array, with the cells to modify set to -2
36 ibound = reshapem2v(matload('init_bnds.inf'))
37
38 h = floatload('init_head_orig.mod')
39
40 # these are relative coordinates, -1 <= x,y < +1
41 x = floatload('rel_x_coord.dat')
42 y = floatload('rel_y_coord.dat')
43
44 # unpack surface parameters (one per line)
45 # z = A + B*(y + D*sign(y)*sqrt(abs(y)))+C*(E*x**3 - F*x**2 - x)
46
47 finput = open('surface_par_params.in', 'r')
48 try:
49     a,b,c,d,e,f,exp = [float(line.rstrip()) for line in finput]
```

```

50 except ValueError:
51     # python doesn't like 'D' in 1.2D-4 notation used by PEST sometimes.
52     finput.seek(0)
53     lines = [line.rstrip() for line in finput]
54     for i in range(len(lines)):
55         lines[i] = lines[i].replace('D', 'E')
56     a,b,c,d,e,f,exp = [float(line) for line in lines]
57
58 finput.close()
59
60 # file to output initial/boundary head for MODFLOW model
61 fout = open('init_head.mod', 'w')
62 for i in xrange(len(ibound)):
63     if ibound[i] == '-2' or ibound[i] == '1':
64         # apply exponential surface to active cells (ibound=1) -> starting guess
65         # and non-geologic boundary conditions (ibound=-2) -> constant head value
66         if y[i] == 0:
67             fout.write('%.7e \n' % (a + c*(e*x[i]**3 + f*x[i]**2 - x[i])))
68         else:
69             fout.write('%.7e \n' % (a + b*(y[i] + d*sign(y[i])*abs(y[i])**exp) +
70                                     c*(e*x[i]**3 + f*x[i]**2 - x[i])))
71     else:
72         # use land surface at constant head east of halite boundary
73         # ibound=0 doesn't matter (inactive)
74         fout.write('%.7e\n' % h[i])
75
76 fout.close()

```

6.3.6 Bash shell script run_02_model

```
1 #!/bin/bash
2
3 #set -o xtrace
4
5 #echo 'step 1: surface extrapolate'
6 python surface_02_extrapolate.py
7
8 # run modflow
9 #echo 'step 2: run modflow'
10 ../bin/mf2k/mf2k_1.6.release mf2k_head.nam >/dev/null
11
12 # run mod2obs
13 #echo 'step 3: extract observations'
14 ../bin/Builds/Linux/mod2obs.exe < mod2obs_head.in >/dev/null
15
16 # create meta-observations of N vs. S
17 python create_average_NS_residuals.py
```


6.3.7 Python script head_bin2ascii.py

```
1 import struct
2 from sys import argv, exit
3
4 class FortranFile(file):
5     """ modified from May 2007 Enthought-dev mailing list post by Neil Martinsen-Burrell """
6
7     def __init__(self, fname, mode='r', buf=0):
8         file.__init__(self, fname, mode, buf)
9         self.ENDIAN = '<' # little endian
10        self.di = 4 # default integer (could be 8 on 64-bit platforms)
11
12    def readReals(self, prec='f'):
13        """Read in an array of reals (default single precision) with error checking"""
14        # read header (length of record)
15        l = struct.unpack(self.ENDIAN+'i', self.read(self.di))[0]
16        data_str = self.read(l)
17        len_real = struct.calcsize(prec)
18        if l % len_real != 0:
19            raise IOError('Error reading array of reals from data file')
20        num = l/len_real
21        reals = struct.unpack(self.ENDIAN+str(num)+prec, data_str)
22        # check footer
23        if struct.unpack(self.ENDIAN+'i', self.read(self.di))[0] != 1:
24            raise IOError('Error reading array of reals from data file')
25        return list(reals)
26
27    def readInts(self):
28        """Read in an array of integers with error checking"""
29        l = struct.unpack('i', self.read(self.di))[0]
30        data_str = self.read(l)
31        len_int = struct.calcsize('i')
32        if l % len_int != 0:
33            raise IOError('Error reading array of integers from data file')
34        num = l/len_int
35        ints = struct.unpack(str(num)+'i', data_str)
36        if struct.unpack(self.ENDIAN+'i', self.read(self.di))[0] != 1:
37            raise IOError('Error reading array of integers from data file')
38        return list(ints)
39
40    def readRecord(self):
41        """Read a single fortran record (potentially mixed reals and ints)"""
42        dat = self.read(self.di)
43        if len(dat) == 0:
44            raise IOError('Empty record header')
45        l = struct.unpack(self.ENDIAN+'i', dat)[0]
46        data_str = self.read(l)
47        if len(data_str) != l:
48            raise IOError('Didn't read enough data')
49        check = self.read(self.di)
```

```

50         if len(check) != 4:
51             raise IOError('Didn't read enough data')
52         if struct.unpack(self.ENDIAN+'i',check)[0] != 1:
53             raise IOError('Error reading record from data file')
54         return data_str
55
56 def reshapev2m(v,nx,ny):
57     """Reshape a vector that was previously reshaped in C-major order from a matrix,
58     back into a C-major order matrix (here a list of lists)."""
59     m = [None]*ny
60     n = nx*ny
61     for i,(lo,hi) in enumerate(zip(xrange(0, n-nx+1, nx), xrange(nx, n+1, nx))):
62         m[i] = v[lo:hi]
63     return m
64
65 def floatmatsave(filehandle,m):
66     """Writes array to open filehandle, format '568%e12.5'.
67     Outer list is rows, inner lists are columns."""
68
69     for row in m:
70         f.write(''.join([' %12.5e' % col for col in row]) + '\n')
71
72 # open file and set endian-ness
73 try:
74     infn,outfn = argv[1:3]
75 except:
76     print '2 command-line arguments not given, using default in/out filenames'
77     infn = 'modeled_head.bin'
78     outfn = 'modeled_head_asciihed.grd'
79
80 ff = FortranFile(infn)
81
82 # currently this assumes a single-layer MODFLOW model (or at least only one layer of output)
83
84 # format of MODFLOW header in binary layer array
85 fmt = '<2i2f16s3i'
86 # little endian, 2 integers, 2 floats,
87 # 16-character string (4 element array of 4-byte strings), 3 integers
88
89 while True:
90     try:
91         # read in header
92         h = ff.readRecord()
93
94     except IOError:
95         # exit while loop
96         break
97
98     else:
99         # unpack header
100        kstp,kper,pertim,totim,text,ncol,nrow,ilay = struct.unpack(fmt,h)

```

```

101
102     # print status/confirmation to terminal
103     print kstp , kper , pertim , totim , text , ncol , nrow , ilay
104
105     h = ff.readReals()
106
107     ff.close()
108
109     xmin , xmax = (601700.0 , 630000.0)
110     ymin , ymax = (3566500.0 , 3597100.0)
111     hmin = min(h)
112     hmax = max(h)
113
114     # write output in Surfer ASCII grid format
115     f = open(outfn , 'w')
116     f.write( """DSAA
117     %i %i
118     %.1f %.1f
119     %.1f %.1f
120     %.8e %.8e
121     """ %(ncol , nrow , xmin , xmax , ymin , ymax , hmin , hmax) )
122     hmat = reshapev2m(h , ncol , nrow)
123
124     # MODFLOW starts data in upper-left corner
125     # Surfer expects data starting in lower-left corner
126     # flip array in row direction
127
128     floatmatsave(f , hmat[:: -1])
129     f.close()

```

6.3.8 Python script merge_observed_modeled_heads.py

```
1 fobs = open('meas_head_2011ASER.smp','r') # measured head
2 fmod = open('modeled_head.smp','r')     # modeled head
3 fwgt = open('obs_loc_2011ASER.dat','r')  # weights
4 fdb = open('spec_wells.crd','r')        # x/y coordinates
5
6 fout = open('both_heads.smp','w')       # resulting file
7
8 # read in list of x/y coordinates, key by well name
9 wells = {}
10 for line in fdb:
11     well,x,y = line.split()[0:3] # ignore last column
12     wells[well.upper()] = [x,y]
13 fdb.close()
14
15 fout.write('\t'.join(['#NAME','UTM-NAD27-X','UTM-NAD27-Y',
16                      'OBSERVED','MODELED','OBS-MOD','WEIGHT'])+'\n')
17
18 for sobs,smod,w in zip(fobs,fmod,fwgt):
19     obs = float(sobs.split()[3])
20     mod = float(smod.split()[3])
21     name = sobs.split()[0].upper()
22     fout.write('\t'.join([name,wells[name][0],wells[name][1],
23                          str(obs),str(mod),str(obs-mod),
24                          w.rstrip().split()[1]])+'\n')
25
26 fobs.close()
27 fmod.close()
28 fwgt.close()
29 fout.close()
```

6.3.9 Python script `convert_dtrkmf_output_for_surfer.py`

```
1
2 # grid origin for dtrkmf cell -> x,y conversion
3 x0 = 601700.0
4 y0 = 3597100.0
5
6 dx = 100.0
7 dy = 100.0
8
9 fout = open('dtrk_output.blm', 'w')
10
11 # read in all results for saving particle tracks
12 fin = open('dtrk.out', 'r')
13 results = [l.split() for l in fin.readlines()[1:]]
14 fin.close()
15
16 npts = len(results)
17
18 # write Surfer blanking file header
19 fout.write('%i,1\n' % npts)
20
21 # write x,y location and time
22 for pt in results:
23     x = float(pt[1])*dx + x0
24     y = y0 - float(pt[2])*dy
25     t = float(pt[0])/7.75*4.0 # convert to 4m Cuelbra thickness
26     fout.write('%%.1f,%.1f,%.8e\n' % (x,y,t))
27
28 fout.close()
```

6.3.10 Python script plot-contour-maps.py

```
1 import numpy as np
2 #import matplotlib
3 #matplotlib.use('Agg')
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.basemap import pyproj
6
7 # http://spatialreference.org/ref/epsg/26713/
8 # http://spatialreference.org/ref/epsg/31013/
9 putm = pyproj.Proj(init='epsg:26713') # UTM Zone 13N NAD27 (meters)
10 pstp = pyproj.Proj(init='epsg:32012') # NM state plane east NAD27 (meters)
11
12 def transform(xin, yin):
13     """does the default conversion from utm -> state plane
14     then also convert to feet from meters"""
15     xout, yout = pyproj.transform(putm, pstp, xin, yin)
16     xout /= M2FT
17     yout /= M2FT
18     return xout, yout
19
20 year = '2011'
21 fprefix = 'pest_02/'
22 mprefix = '../wipp-polyline-data/'
23 cfname = fprefix + 'modeled_head_pest_02.grd'
24 pfname = fprefix + 'dtrk_output_pest_02.blm'
25 wfname = fprefix + 'modeled_vs_observed_head_pest_02.txt'
26
27 M2FT = 0.3048
28
29 # read in well-related things
30 # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31 # load in observed, modeled, obs-mod, (all in meters)
32 res = np.loadtxt(wfname, skiprows=1, usecols=(3,4,5))
33 res /= M2FT # convert heads to feet
34 wellx, welly = transform(*np.loadtxt(wfname, skiprows=1, usecols=(1,2), unpack=True))
35 names = np.loadtxt(wfname, skiprows=1, usecols=(0,), dtype='|S6')
36
37 # read in head-related things
38 # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 h = np.loadtxt(cfname, skiprows=5) # ASCII matrix of modeled head in meters AMSL
40 h[h<0.0] = np.NaN # no-flow zone in northeast
41 h[h>1000.0] = np.NaN # constant-head zone in east
42 h /= M2FT # convert elevations to feet
43
44 # surfer grid is implicit in header
45 # create grid from min/max UTM NAD27 coordinates (meters)
46 utmy, utmx = np.mgrid[3566500.0:3597100.0:307j, 601700.0:630000.0:284j]
47
48 # head contour coords
49 hx, hy = transform(utmx, utmy)
```

```

50 del utmx, utmy
51
52 # read in particle-related things
53 # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
54 px,py = transform(*np.loadtxt(pfname, skiprows=1, delimiter=',', usecols=(0,1), unpack=True))
55 part = np.loadtxt(pfname, skiprows=1, delimiter=',', usecols=(2,))
56
57 # read in MODFLOW model, WIPP LWB & ASER contour domain (UTM X & Y)
58 # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59 modx,mody = transform(*np.loadtxt(mprefix+'total_boundary.dat', unpack=True))
60 wippx, wippy = transform(*np.loadtxt(mprefix+'wipp_boundary.dat',
61                                usecols=(0,1), unpack=True))
62 aserx, asery = transform(*np.loadtxt(mprefix+'ASER_boundary.csv',
63                                delimiter=',', usecols=(1,2), unpack=True))
64
65 a = []
66
67 # plot contour map of entire model area
68 # *****
69 fig = plt.figure(1, figsize=(12,16))
70 ax = fig.add_subplot(111)
71 lev = 3000 + np.arange(17)*10
72 CS = ax.contour(hx,hy,h, levels=lev, colors='k', linewidths=0.5)
73 ax.clabel(CS, lev[:,2], fmt='%i')
74 ax.plot(wippx, wippy, 'k-')
75 ax.plot(aserx, asery, 'g-')
76 ax.plot(modx,mody, '- ', color='purple', linewidth=2)
77 ax.plot(wellx, welly, linestyle='none', marker='o',
78         markeredgecolor='green', markerfacecolor='none')
79 ax.set_xticks(630000 + np.arange(10.0)*10000)
80 ax.set_yticks(450000 + np.arange(10.0)*10000)
81 labels = ax.get_yticklabels()
82 for label in labels:
83     label.set_rotation(90)
84 for x,y,n in zip(wellx, welly, names):
85     # plot just above
86     a.append(plt.annotate(n, xy=(x,y), xytext=(0,5),
87                          textcoords='offset points',
88                          horizontalalignment='center',
89                          fontsize=8))
90 plt.axis('image')
91 ax.set_title('Freshwater Heads Model Area '+year)
92 ax.set_xlabel('NAD27 NM East State Plane Easting (ft)')
93 ax.set_ylabel('NAD27 NM East State Plane Northing (ft)')
94
95 # compute travel time and path length to WIPP LWB
96 # *****
97
98 # compute incremental distance between times
99 pd = M2FT*np.sqrt((px[1:] - px[:-1])**2 + (py[1:] - py[:-1])**2)
100

```



```

152         textcoords='offset points',
153         horizontalalignment='center',
154         fontsize=6))
155 ax.set_title('Freshwater Heads WIPP Area '+ year)
156 ax.set_xlabel('NAD27 NM East State Plane Easting (ft)')
157 ax.set_ylabel('NAD27 NM East State Plane Northing (ft)')
158
159 ### >>>>>>>manually fix labels>>>>
160 for lab in a:
161     lab.draggable()
162 plt.show()
163 ### <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
164
165 plt.savefig('aser-area-contour-map'+year+'.png')
166 plt.close(1)

```

6.3.11 Python script plot-results-bar-charts.py

```
1 import numpy as np
2 import matplotlib
3 matplotlib.use('Agg')
4 import matplotlib.pyplot as plt
5
6 fprefix = 'pest_02/'
7 mprefix = '../..//wipp-polyline-data/'
8 fname = fprefix + 'modeled_vs_observed_head_pest_02.txt'
9
10 ofname = 'original_average/modeled_vs_observed_head_original_average.txt'
11
12 M2FT = 0.3048
13 year = '2011'
14
15 # load in observed, modeled, obs-mod, (all in meters)
16 res = np.loadtxt(fname, skiprows=1, usecols=(3,4,5))
17 ores = np.loadtxt(ofname, skiprows=1, usecols=(3,4,5))
18
19 # load in weights
20 weights = np.loadtxt(fname, skiprows=1, usecols=(6,), dtype='int')
21 # load in names
22 names = np.loadtxt(fname, skiprows=1, usecols=(0,), dtype='|S6')
23
24 # load in N/S/C/X zones
25 zones = np.loadtxt('obs_loc_%sASER.dat' % year, usecols=(2,), dtype='|S1')
26
27 ## checking locations / zones
28 # *****
29 wipp = np.loadtxt(mprefix+'wipp_boundary.dat')
30 x,y = np.loadtxt(fname, skiprows=1, usecols=(1,2), unpack=True)
31
32 fig = plt.figure(2, figsize=(18,12))
33 ax1 = fig.add_subplot(121)
34 ax1.plot(x,y, 'k*') # wells
35 ax1.plot(wipp[:,0], wipp[:,1], 'r-') # WIPP LWB
36 buff = np.loadtxt(mprefix+'wipp_boundary.dat')
37 buff[1:3,0] -= 3000.0
38 buff[0,0] += 3000.0
39 buff[3:,0] += 3000.0
40 buff[2:4,1] -= 3000.0
41 buff[0:2,1] += 3000.0
42 buff[-1,1] += 3000.0
43 colors = {'N':'red', 'S':'blue', 'C':'green', 'X':'gray'}
44 ax1.plot(buff[:,0], buff[:,1], 'g--') # WIPP LWB+3km
45 for xv,yv,n,w,z in zip(x,y,names,weights,zones):
46     print xv,yv,n,w,z
47     plt.annotate('%s %i'%(n,w), xy=(xv,yv), fontsize=8, color=colors[z])
48 plt.axis('image')
49 ax1.set_xlim([x.min()-1000,x.max()+1000])
```

```

50 ax1.set_ylim([y.min()-1000,y.max()+1000])
51 ax2 = fig.add_subplot(122)
52 ax2.plot(x,y, 'k*') # wells
53 ax2.plot(wipp[:,0], wipp[:,1], 'r-') # WIPP LWB
54 ax2.plot(buff[:,0], buff[:,1], 'g--') # WIPP LWB+3km
55 for xv,yv,n,w,z in zip(x,y,names,weights,zones):
56     plt.annotate('%s %i' % (n,w),xy=(xv,yv),fontsize=8,color=colors[z])
57 plt.axis('image')
58 ax2.set_xlim([wipp[:,0].min()-100,wipp[:,0].max()+100])
59 ax2.set_ylim([wipp[:,1].min()-100,wipp[:,1].max()+100])
60 plt.suptitle('well weights check '+year)
61 plt.savefig('check-well-weights-'+year+'.png')
62
63 # convert lengths to feet
64 res /= M2FT
65 ores /= M2FT
66
67 # create the histogram of residuals for ASER
68 # *****
69
70 # -10,-9,...8,9,10
71 bins = np.arange(-10,11)
72 rectfig = (15,7)
73 squarefig = (8.5,8.5)
74
75 fig = plt.figure(1,figsize=rectfig)
76 ax = fig.add_subplot(111)
77 # all the data, all but distant wells
78 ax.hist([res[weights<2],res[:,2]],bins=bins,range=(-10.0,10.0),
79         rwidth=0.75,align='mid',
80         color=['red','blue'],
81         label=['Inside LWB & <3km from WIPP LWB','All wells'])
82 ax.set_xlabel('Measured-Modeled (ft)')
83 ax.set_ylabel('Frequency')
84 ax.set_xticks(bins)
85 ax.set_ylim([0,10])
86 ax.set_yticks(np.arange(0,10,2))
87 plt.grid()
88 ax.yaxis.grid(True,which='major')
89 ax.xaxis.grid(False)
90 plt.legend(loc='upper left')
91 plt.title('Histogram of Model Residuals '+year)
92 plt.annotate('AEC-7 @ %.1f'%res[0,2],xy=(-9.75,5.0),xytext=(-8.5,5.0),
93            arrowprops={'arrowstyle':'->'},fontsize=16)
94 plt.savefig('model-error-histogram-'+year+'.png')
95 plt.close(1)
96
97 # create bar chart plot of individual residual for ASER
98 # *****
99
100 m0 = weights==0

```

```

101 m1 = weights==1
102 m2 = np.logical_or(weights==2,weights==99)
103
104 # separate wells into groups
105 resin = res[m0,2]
106 resnear = res[m1,2]
107 resfar = res[m2,2]
108
109 nin = resin.size
110 nnear = resnear.size
111 nfar = resfar.size
112
113 # separate names into groups
114 namin = names[m0]
115 namnear = names[m1]
116 namfar = names[m2]
117
118 # get indices that sort vectors
119 ordin = np.argsort(namin)
120 ordnear = np.argsort(namnear)
121 ordfar = np.argsort(namfar)
122
123 # put vectors back together (groups adjacent and sorted inside each group)
124 resagg = np.concatenate((resin[ordin],resnear[ordnear],resfar[ordfar]),axis=0)
125 namagg = np.concatenate((namin[ordin],namnear[ordnear],namfar[ordfar]),axis=0)
126
127 fig = plt.figure(1,figsize=rectfig)
128 ax = fig.add_subplot(111)
129
130 wid = 0.6
131 shift = 0.5 - wid/2.0
132 ab = np.arange(res.shape[0])
133
134 print ab.shape
135 print ab
136
137 ax.bar(left=ab+shift,height=resagg,width=0.6,bottom=0.0,color='gray')
138 ax.set_ylim([-15.0,15.0])
139 ax.spines['bottom'].set_position('zero')
140 ax.spines['top'].set_color('none')
141 ax.xaxis.set_ticks_position('bottom')
142 plt.xticks(ab+wid,namagg,rotation=90)
143 # vertical lines dividing groups
144 ax.axvline(x=nin,color='black',linestyle='dashed')
145 ax.axvline(x=nin+nnear,color='black',linestyle='dashed')
146 ax.axhline(y=0,color='black',linestyle='solid')
147 ax.axhline(y=-15,color='black',linestyle='dotted')
148 plt.grid()
149 ax.yaxis.grid(True,which='major')
150 ax.xaxis.grid(False)
151 ax.set_xlim([0,res.shape[0]])

```

```

152
153 plt.annotate('',xy=(0.0,12.0),xycoords='data',
154             xytext=(nin,12.0),textcoords='data',
155             arrowprops={'arrowstyle':'<->'})
156 plt.annotate('inside WIPP LWB',xy=(nin/3.0,12.5),xycoords='data')
157
158 plt.annotate('',xy=(nin,12.0),xycoords='data',
159             xytext=(nin+nnear,12.0),textcoords='data',
160             arrowprops={'arrowstyle':'<->'})
161 plt.annotate('<3km WIPP LWB',xy=(nin+nnear/3.0,12.5),xycoords='data')
162
163 plt.annotate('',xy=(nin+nnear,12.0),xycoords='data',
164             xytext=(nin+nnear+nfar,12.0),textcoords='data',
165             arrowprops={'arrowstyle':'<->'})
166 plt.annotate('>3km WIPP LWB',xy=(nin+nnear+nfar/3.0,12.5),xycoords='data')
167
168 ax.set_ylabel('Measured-Modeled (ft)')
169 ax.set_title('individual residuals '+year)
170 plt.annotate('AEC-7 @ %.1f'%res[0,2],xy=(nin+nnear+1.0,-14.5),xycoords='data')
171
172 plt.savefig('model-error-residuals-'+year+'.png')
173 plt.close(1)
174
175
176 # create scatter plot of measured vs. modeled
177 # *****
178 m = 1.0/M2FT
179 sr = [2980,3120]
180
181
182 print 'modeled-vs-measured correlation coefficients'
183 print 'all data: %.4f' % np.corrcoef(res[:,0],res[:,1])[1,0]**2
184 print 'inside WIPP: %.4f' % np.corrcoef(res[weights==0,0],res[weights==0,1])[1,0]**2
185 print 'inside 3km: %.4f' % np.corrcoef(res[weights<2,0],res[weights<2,1])[1,0]**2
186
187 print 'uncalibrated model'
188 print 'all data: %.4f' % np.corrcoef(ores[:,0],ores[:,1])[1,0]**2
189 print 'inside WIPP: %.4f' % np.corrcoef(ores[weights==0,0],ores[weights==0,1])[1,0]**2
190 print 'inside 3km: %.4f' % np.corrcoef(ores[weights<2,0],ores[weights<2,1])[1,0]**2
191
192 fig = plt.figure(1,figsize=squarefig)
193 ax = fig.add_subplot(111)
194 ax.plot(res[m0,0],res[m0,1],color='red',markersize=10,
195         marker='+',linestyle='none',label='Inside LWB')
196 ax.plot(res[m1,0],res[m1,1],color='green',markersize=10,
197         marker='x',linestyle='none',label='< 3km From LWB')
198 ax.plot(res[m2,0],res[m2,1],color='blue',markersize=10,
199         marker='*',linestyle='none',label='distant')
200 ax.plot(sr,sr,'k-',label='$45^\{\degree\}$ Perfect Fit')
201 ax.plot([sr[0],sr[1]],[sr[0]+m,sr[1]+m],'g-',linewidth=0.5,label='$\pm$ 1m Misfit')
202 ax.plot([sr[0],sr[1]],[sr[0]-m,sr[1]-m],'g-',linewidth=0.5,label='__nolegend__')

```

```

203 ax.set_xticks(np.linspace(sr[0], sr[1], 8))
204 ax.set_yticks(np.linspace(sr[0], sr[1], 8))
205 ax.set_xlim(sr)
206 ax.set_ylim(sr)
207 plt.minorticks_on()
208 plt.legend(loc='lower right', scatterpoints=1, numpoints=1)
209 plt.grid()
210 for j, lab in enumerate(names):
211     if res[j, 2] < -1.5*m:
212         # plot labels to left of value far above 45-degree line
213         plt.annotate(lab, xy=(res[j, 0], res[j, 1]),
214                     xytext=(res[j, 0] - (2.9*len(lab)), res[j, 1] - 2.0), fontsize=14)
215     elif res[j, 2] > 1.5*m:
216         # plot labels to right of value far below 45-degree line
217         plt.annotate(lab, xy=(res[j, 0], res[j, 1]),
218                     xytext=(res[j, 0] + 2.0, res[j, 1] - 2.0), fontsize=14)
219 ax.set_xlabel('Observed Freshwater Head (ft AMSL)')
220 ax.set_ylabel('Modeled Freshwater Head (ft AMSL)')
221 ax.set_title('modeled vs. measured '+year)
222 plt.savefig('scatter_pest_02_'+year+'.png')

```