562110

Analysis Report for Preparation of 2013 Culebra Potentiometric Surface Contour Map

Task Number: 4.4.2.3.1

Report Date: 6/10/2014

Author: Kristopher L. Kuhlman, 6224 — 6/10/14 Date
Applied Systems Analysis & Research Department

Technical Review: Todd Zeitler, 6211 — 6/10/2014 Date
Performance Assessment Department

QA Review: Shelly R. Nielsen, 6210 — 6-10-14 Date
Carlsbad Programs Group

Management Review: Christi D. Leigh, 6212 — 6-10-14 Date
Manager, Repository Performance Department

WIPP:4.4.2.3.1:TD:QA-L:RECERT:549085

Table of Contents

1 Introduction

This report documents the preparation of the 2013 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). 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 Sandia National Laboratories procedure SP 9-9 (Kuhlman, 2009), which reflects this NMED driver. This report is similar to Kuhlman (2013); the same analysis is performed on data from February 2013, rather than February 2012 data. February 2013 data for contouring were obtained from the WIPP Management & Operations contractor (Watterson, 2014).

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), average parameter fields were used as input to MODFLOW to simulate equivalent freshwater heads within and around the WIPP land withdrawal boundary (LWB). For 2013, PEST is used to adjust a subset of the boundary conditions in the averaged MODFLOW model to improve the match between the observed freshwater heads and the model-predicted heads at Culebra well locations. The output of the averaged, PEST-calibrated MODFLOW model is both contoured and used to compute the 2013 advective particle track forward from the WIPP waste-handling shaft.

2 Scientific Approach

2.1 Modeling Overview

Steady-state groundwater flow simulations were carried out using similar software to what was used for the WIPP Compliance Recertification Application 2009 Performance Assessment Baseline Calculation (CRA-2009 PABC), as presented in the AP-114 Task 7 Analysis Report (Hart et al., 2009), and used in CRA-2014 (DOE, 2014). This setup was used to create the input calibrated fields. See Table 1 for a summary of 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 CRA-2009 PABC and CRA-2014. 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, calibrated using PEST, and used 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 was 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 (excluding AEC-7) and transient drawdown to large-scale pumping tests. Hart et al. (2009) described the calibration effort that went into the CRA-2009 PABC; DOE (2014) summarizes the model development and calibration results. An analogous but much simpler process was used here for the averaged MODFLOW model. PEST was used to modify a subset of the MODFLOW boundary conditions (see red boundaries in Figure 1). For simplicity the boundary conditions were modified (rather than the *T*, *A*, and *R* parameter fields), 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 were the February 2013 measured annual freshwater heads at Culebra monitoring wells. In the averaged MODFLOW model, boundary conditions were 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

Software	Version	Description	Platform	Software QA status
MODFLOW-2000	1.6	Groundwater flow model	PA cluster	Acquired; qualified under NP 19-1 (Harbaugh et al., 2000)
PEST	9.11	Automatic parameter estimation code	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
Python	2.7.3	Scripting language (plotting)	Linux desktop	Commercial off the shelf
Bash	3.00.15	Scripting language (file manipulation)	PA cluster	Commercial off the shelf

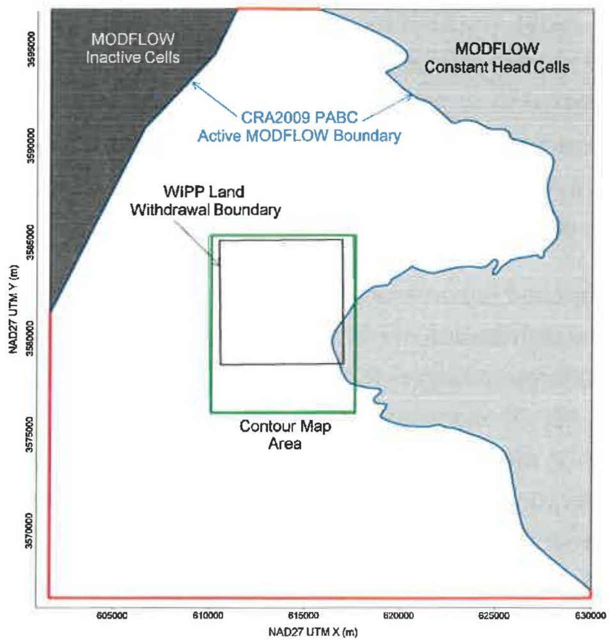

Figure 1. MODFLOW-2000 model domain, adjusted boundary conditions shown in red, contour area outlined in green.

The resulting heads from the PEST-calibrated averaged MODFLOW model were contoured over an area surrounding the WIPP site using matplotlib (a Python plotting library). 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 through the Culebra by a conservative (i.e., non-dispersive and non-reactive) particle from the waste-handling shaft to the WIPP LWB. 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 (a Python array-functionality library), 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 input files and scripts written for this work were executed on the PA Linux cluster (`alice.sandia.gov`), while the creation of figures was done using Python scripts on an Intel-Corei7-equipped desktop computer running Kubuntu Linux, version 12.04.

2.2 Creating Average MODFLOW Simulation

An averaged MODFLOW model is used to compute the equivalent freshwater head and cell-by-cell flow solution. The computed heads are contoured and the flow solution is 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 average 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 physics embodied in the MODFLOW simulator code). An alternative approach would average outputs from 100 models to produce a single average result, but average 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 all 100 realizations). This simplification results in "smooth" freshwater head contours and relatively faster particle tracks, compared to those predicted by the any one of the 100 fields calibrated as part of AP114 Task 7 (Hart et al., 2009).

The MODFLOW model grid is a single 7.5-m thick layer, comprising 307 rows and 284 columns; each model cell is 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 north.

The calibrated *T*, *A*, and *R* parameter fields from the PA MODFLOW model were checked out of the PA version control 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 `/nfs/data/CVSLIB/Analyses/SP9_9`). Model inputs can be divided into two groups. The first group includes model inputs that are common 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 of the 100 realizations. The constant model inputs in the first group are used directly in the averaged MODFLOW model, 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 geometrically averaged (i.e., the arithmetic average was computed 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 boundary condition calibration proceeds. There are two types of boundary conditions in the WIPP MODFLOW model. 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 this 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 straight-line southern, southwestern, and northern boundaries that coincide with the primary compass directions and the rectangular frame surrounding the model domain (shown as heavy red lines in Figure 1). The value of specified head assigned in boundary cells corresponding 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) with 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 no-flow constant head region to the east of WIPP. 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 Python scripts elsewhere in this analysis).

Using the output from `boundary_types.py`, the Python script `surface_02_extrapolate.py` computes the initial head at active model cells (IBOUND=1) and the specified constant-head at the adjustable 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 two major types of inputs to PEST. The first input class is the "forward model", which 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 PEST runs repeatedly to estimate sensitivities of model outputs to model inputs. The second 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 from February 2013 (H-09bR used a January freshwater head, to avoid abnormally high water levels in February) used as targets for the PEST calibration from Watterson (2014) are listed in Table 2, and specified along with weights in the PEST configuration files. SNL-13 was indicated in Waterson (2014) as being an anomalous level, which should be excluded from mapping. Excluding this datapoint from the analysis resulted in high predicted water levels in all the wells on the southern portion of the WIPP site. Scientific judgment was used to include the well in the contour map generation exercise.

Table 2. Freshwater Head Calibration Targets used in PEST, from Watterson (2014).

Well	Measurement Date	Freshwater Head Elevation (m AMSL)	Culebra Groundwater Density (g/cm^3)
AEC-7	02/08/13	933.39	1.067
C-2737	02/12/13	920.45	1.023
ERDA-9	02/12/13	924.54	1.073
H-02b2	02/12/13	928.05	1.012
H-03b2	02/12/13	918.01	1.036
H-04bR	02/11/13	916.38	1.017
H-05b	02/08/13	939.65	1.095
H-06bR	02/11/13	935.97	1.038
H-07b1	02/07/13	913.92	1.007
H-09bR	01/07/13	912.99	1.000
H-10c	02/08/13	923.85	1.094
H-11b4R	02/11/13	916.61	1.076
H-12	02/11/13	918.46	1.113
H-15R	02/12/13	919.66	1.118
H-16	02/12/13	928.42	1.037
H-17	02/08/13	916.55	1.133
H-19b0	02/11/13	918.51	1.066
IMC-461	02/07/13	926.97	1.000
SNL-01	02/07/13	938.85	1.029
SNL-02	02/07/13	936.48	1.009
SNL-03	02/07/13	938.41	1.028
SNL-05	02/07/13	936.57	1.009
SNL-08	02/08/13	930.75	1.094
SNL-09	02/07/13	930.82	1.018
SNL-10	02/07/13	930.60	1.009
SNL-12	02/08/13	915.02	1.006
SNL-13	02/07/13	918.62	1.018
SNL-14	02/08/13	915.65	1.046
SNL-16	02/07/13	917.54	1.009
SNL-17	02/08/13	916.02	1.005
SNL-18	02/07/13	936.49	1.005
SNL-19	02/07/13	936.49	1.007
WIPP-11	02/12/13	939.03	1.038
WIPP-13	02/12/13	937.58	1.041
WIPP-19	02/12/13	933.54	1.052
WQSP-1	02/12/13	937.42	1.051
WQSP-2	02/12/13	939.53	1.048
WQSP-3	02/12/13	936.70	1.147
WQSP-4	02/11/13	919.07	1.077
WQSP-5	02/12/13	918.27	1.027
WQSP-6	02/08/13	921.77	1.015

To minimize the number of estimable parameters, and to ensure a degree of smoothness in the specified 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:

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

where $\operatorname{abs}(y)$ is absolute value and $\operatorname{sign}(y)$ 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 $A = 928$, $B = 8$, $C = 1.2$, $D = 1$, $E = 1$, $F = -1$ and $\alpha = 0.5$ are used with the above equation to assign the boundary conditions in the PA MODFLOW model.

PEST was 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 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 MODFLOW output), the template files (how to write the MODFLOW input), and the PEST control file (listing the ranges and initial values for the estimable parameters and the values and weights associated with observations). The wells used in each year's PEST calibration were separated into three groups. Higher observation weights (2.5) were assigned to wells inside the LWB, and lower observation weights (0.4) were assigned to wells distant to the WIPP site, while wells in the near the WIPP LWB 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 far from the WIPP LWB and 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 results. 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`.

3 2013 Results

3.1 2013 Equivalent 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 to the right of the purple line 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.

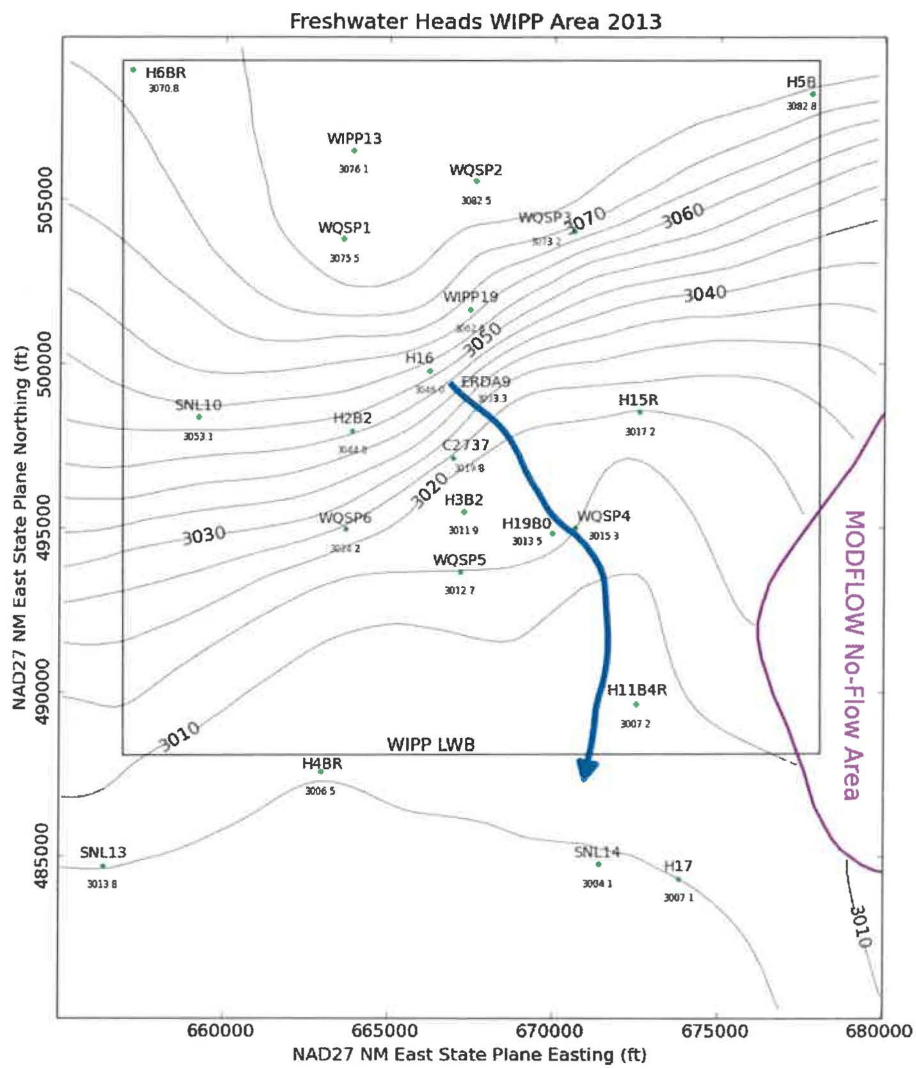

Figure 2. Model-generated February 2013 freshwater head contours with observed head listed at each well (5-foot contour interval) with blue water particle track from waste handling shaft to WIPP LWB. Purple curve is Rustler halite margin.

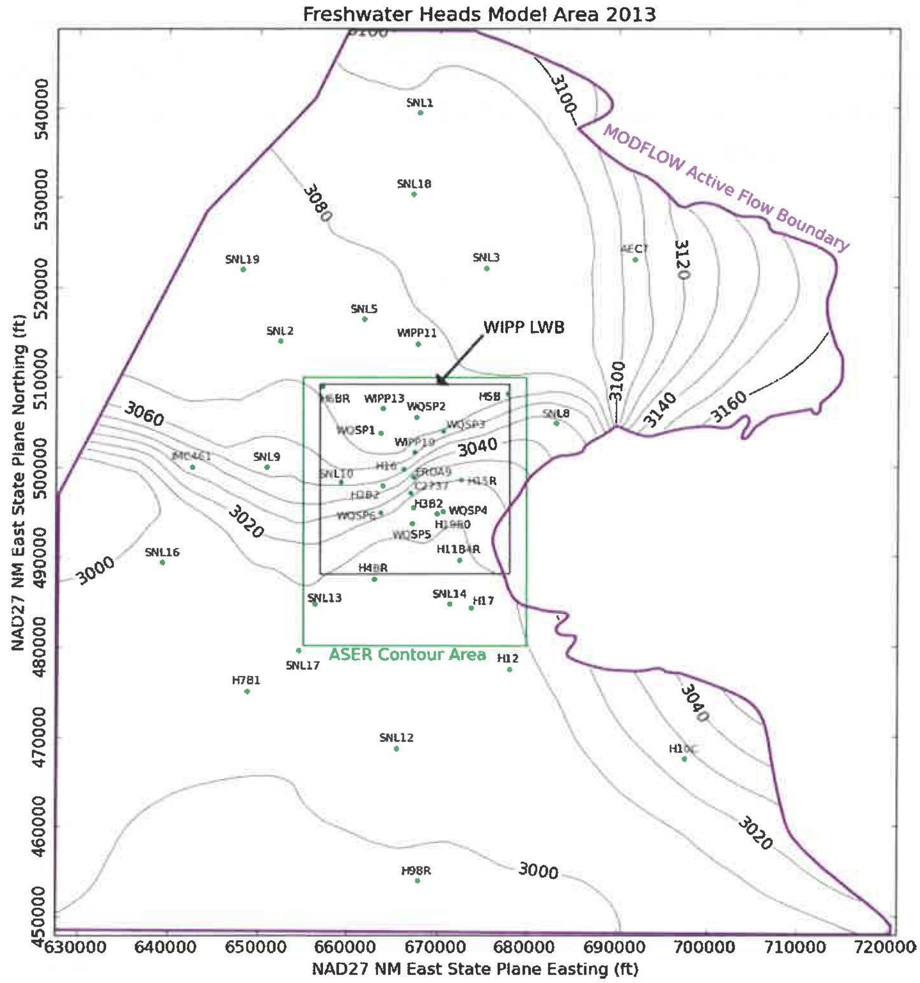

Figure 3. MODFLOW-modeled February 2013 heads for entire model domain (10-foot contour interval). Green rectangle indicates region contoured in Figure 2, black square is WIPP LWB.

3.2 2013 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 4073 m). Assuming the transmissive portion of the Culebra is only 4-m thick, and assuming a constant porosity of 16%, the travel time to the WIPP LWB is 6234 years (output from DTRKMF is adjusted from an original 7.75-m Culebra thickness). This is an average velocity of 0.65 m/yr.

3.3 2013 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 indicated with blue stars. 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).

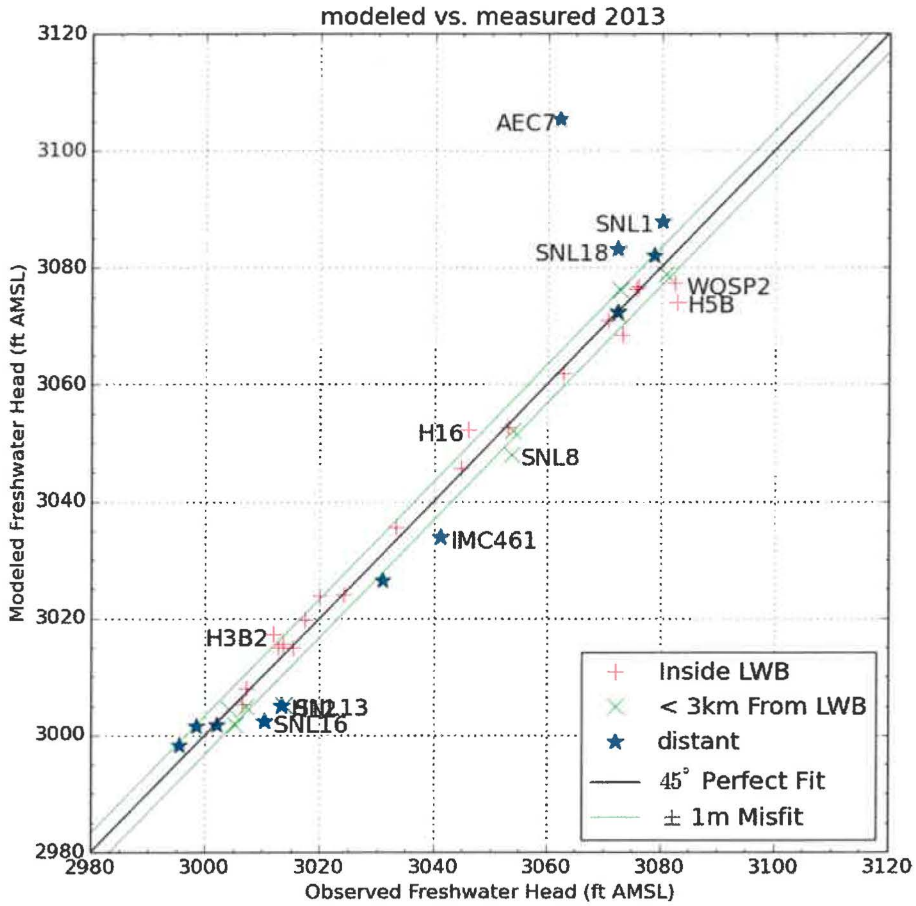

Figure 4. Measured vs. modeled scatter plot for averaged MODFLOW model generated heads and February 2013 observed freshwater heads

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 has a large misfit (13 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 calibrated parameters (for equation 1) were A = 929.4, B = 8.76, C = -1.036, D = 0.7920, E = -0.8611, F = -2.340, and α = -0.4502. The parameters α (exponent on y), C (coefficient on all x variability), and E (coefficient on x^3 variability) had the largest relative change (~186-190%) compared to the starting values. Parameter F (coefficient on x^2) was within -134% of its original value, and D (coefficient on exponentiated y term) was 21% away. All other parameters were <10% different from 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 98%, and the calibration decreased the R^2 value very slightly when including all wells. The calibration improved the fit for the wells in and near the WIPP LWB at the expense of fit to wells distant from the LWB. The distribution of residuals in Figure 5 does not have a strong bias.

Table 3. 2013 Measured vs. Modeled correlation coefficients

	dataset	measured vs. modeled R^2
Uncalibrated	wells inside WIPP LWB	0.985
	wells <3km from WIPP LWB	0.983
	all wells	0.942
Calibrated	wells inside WIPP LWB	0.988
	wells <3km from WIPP LWB	0.985
	all wells	0.938

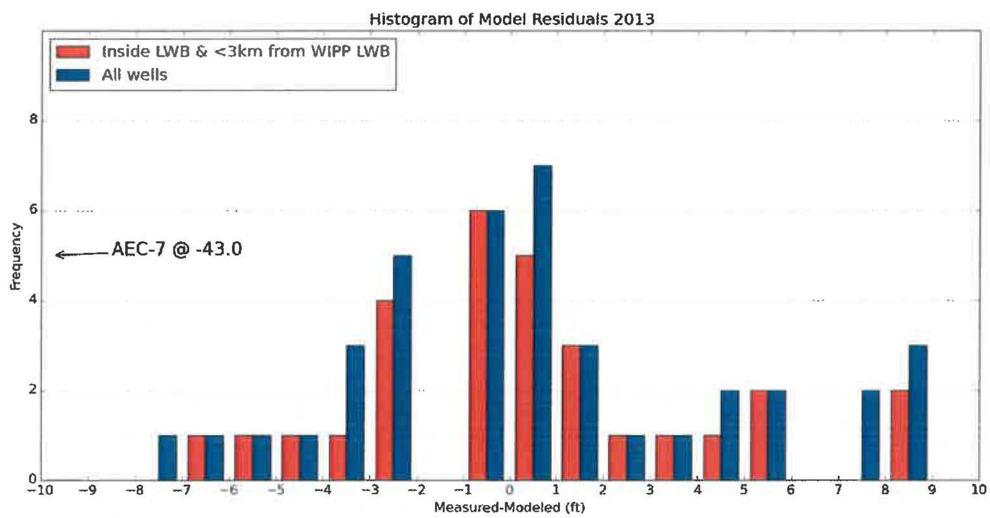

Figure 5. Histogram of Measured-Modeled errors for 2013

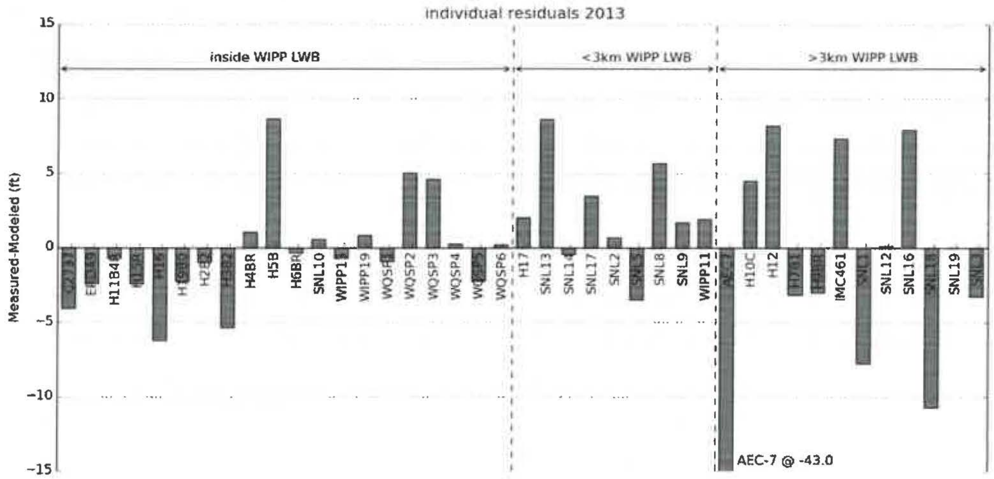

Figure 6. Measured-Modeled errors at each well location for 2013

Aside from the poor fit at AEC-7, the model fit to the February 2013 observations is good. The residual at SNL-18 is over 10 feet, because the observed water level at this well has seen significant fluctuations due to oil and gas activity. PEST is not able to match these observed variations through changes to the boundary conditions. The averaged MODFLOW model captures the bulk Culebra flow behavior, while the PEST calibration improved the model fit to the specific February 2013 observations.

4 References

DOE (US Department of Energy). 2014. *Title 40 CFR Part 191 Subparts B and C Compliance Recertification Application 2014 for the Waste Isolation Pilot Plant Appendix TFIELD-2014 Transmissivity Fields*. US Department of Energy: Carlsbad, NM, DOE/WIPP-14-3503.

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*. Sandia National Laboratories: Carlsbad, NM, ERMS 552391.

Kuhlman, K.L. 2013. Analysis Report for Preparation of 2012 Culebra Potentiometric Surface Contour Map, Sandia National Laboratories: Carlsbad, NM, ERMS 560306.

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. 2014. February 2013 Culebra ASER map data, Washington TRU Solutions, Carlsbad, NM. WIPP Records Center, ERMS 562028.

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 3 parameter fields (transmissivity (*T*), anisotropy (*A*), and recharge (*R*), and storativity (*S*)) from CVS version control 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 geometric average at each cell across the 100 realizations. The 400 input files are each saved as flattened matrices, in row-major order. The average result is saved into 4 parameter files, 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`.

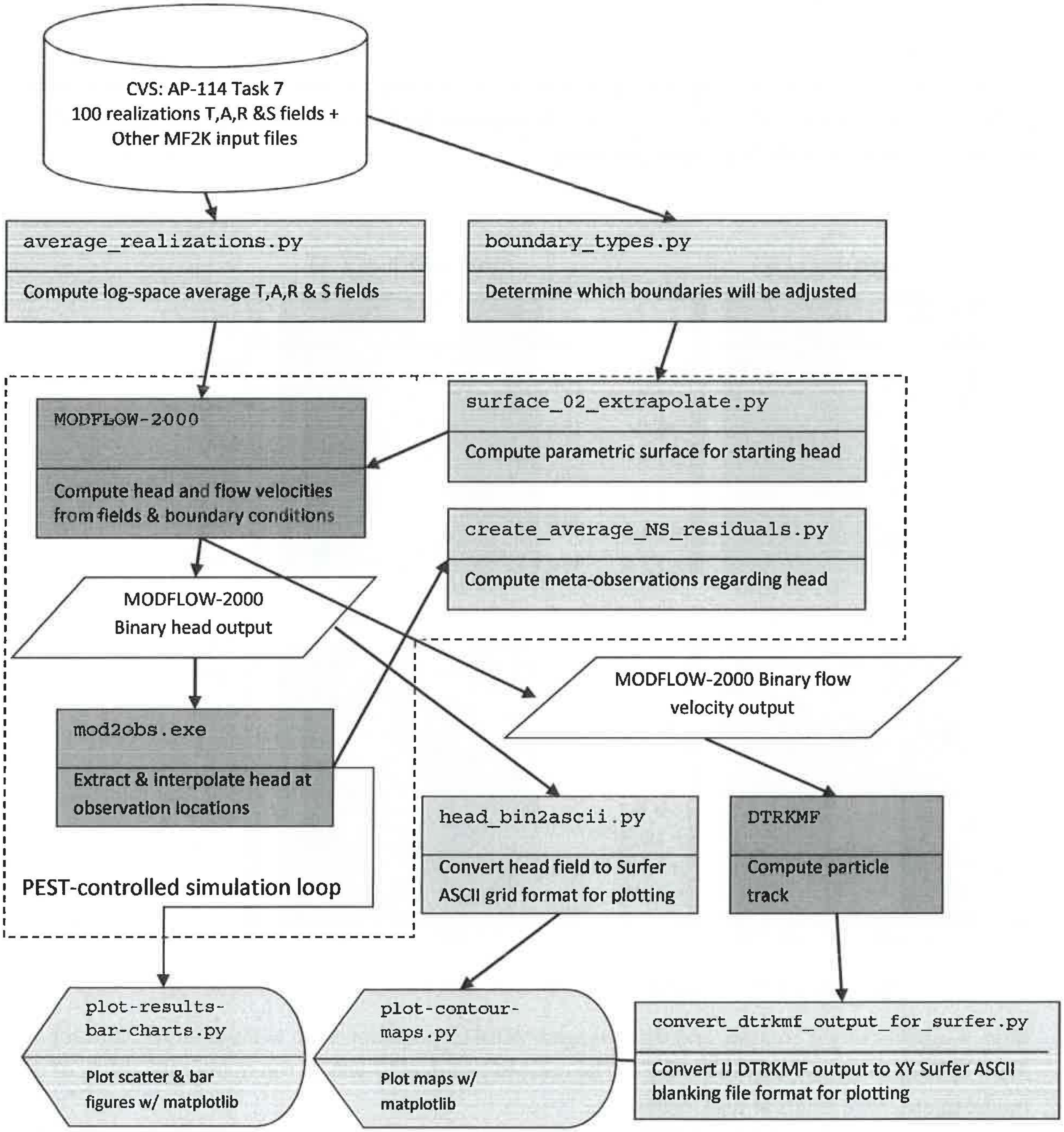

Figure 7. Process flowchart; dark gray indicates qualified programs, light gray are scripts written for this analysis

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 $\log_{10}$ parameters, which compare with similar figures in Hart et al. (2009); inactive regions (< 10^{-15}) were reset to 1 to improve the plotted color scale. The rest of the calculations are done with these averaged fields.

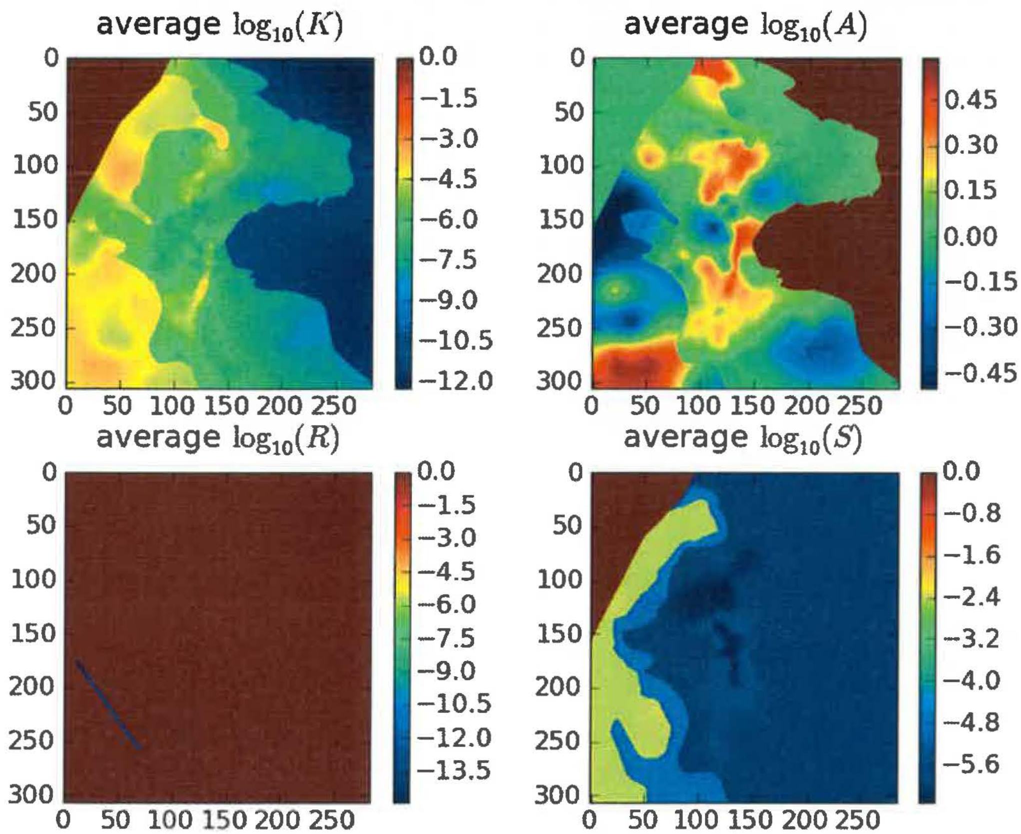

Figure 8. Plots of base-10 logarithms of average parameter fields; rows and columns are labeled on edges of figures.

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 7.75 m to 4 m. 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 than the original cumulative travel time by the factor 7.75/4=1.9375). The conversion from row, column to *x*, *y* is

$$X = 601700 + 100 * column$$
$$Y = 3597100 - 100 * row \quad ,$$

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 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
 0.00000000E+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, 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 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_2013ASER.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_2013ASER.smp`), and the weights are read in from the input file (`obs_loc_2013ASER.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 Files and Script Source Listings

6.1 Input Files

bytes	file type	description	file name
1.5K	Python script	average 100 realizations	`average_realizations.py`
2.1K	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`
624	Python script	convert DTRKMF IJ output to Surfer X,Y blanking format	`convert_dtrkmf_output_for_surfer.py`
2.8K	Python script	create meta observations of avg heat	`create_average_NS_residuals.py`
3.1K	Python script	create PEST input files from observed data	`create_pest_02_input.py`
48	input listing	responses to DTRKMF prompts	`dtrkmf.in`
4.0K	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 2013 heads in `mod2obs.exe` bore sample file format	`meas_head_2013ASER.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`
393	input	listing of wells and geographic groupings	`obs_loc_2013ASER.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`
1.8K	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.bln
16K	PEST output	matrix condition numbers	bc_adjust_2013ASER.cnd
2.7K	PEST output	binary intermediate file	bc_adjust_2013ASER.drf
7.4K	PEST output	binary intermediate file	bc_adjust_2013ASER.jac
7.5K	PEST output	binary intermediate file	bc_adjust_2013ASER.jco
9.9K	PEST output	binary intermediate file	bc_adjust_2013ASER.jst
3.8K	PEST output	parameter statistical matrices	bc_adjust_2013ASER.mtt
477	PEST output	parameter file	bc_adjust_2013ASER.par
62K	PEST output	optimization record	bc_adjust_2013ASER.rec
4.6K	PEST output	model outputs for last iteration	bc_adjust_2013ASER.rei
8.4K	PEST output	summary of residuals	bc_adjust_2013ASER.res
28	PEST output	binary restart file	bc_adjust_2013ASER.rst
24K	PEST output	relative parameter sensitivities	bc_adjust_2013ASER.sen
4.0K	PEST output	absolute parameter sensitivities	bc_adjust_2013ASER.seo
213K	png image	matplotlib plot (Fig. 2)	aser-area-contour-map2013.png
223K	png image	matplotlib plot (Fig. 3)	large-area-contour-map2013.png
33K	png image	matplotlib plot (Fig. 5)	model-error-histogram2013.png
55K	png image	matplotlib plot (Fig. 6)	model-error-residuals2013.png
93K	png image	matplotlib plot (Fig. 4)	scatter_pest_02_2013.png

Table 2: Listing of Output Files

6.3 Individual scripts

6.3.1 Bash shell script checkout_average_run_modflow.sh

```bash
#!/bin/bash

set -o nounset # explode 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)

set -o xtrace

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_2013ASER.smp .
ln -sf ../obs_loc_2013ASER.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`
```

```bash

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_2013ASER

    # 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

```
from math import log10,pow

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

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')

# 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

```
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 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

```python
prefix = '2013ASER'

##################################################
## 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()
```

```python


#####################
## 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
```

```python
    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
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
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] != 1:
            raise IOError('Error reading record from data file')
        return data_str

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 outpu

# 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`

```python
fobs = open('meas_head_2013ASER.smp','r') # measured head
fmod = open('modeled_head.smp','r')        # modeled head
fwgt = open('obs_loc_2013ASER.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

# 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 Culebra thickness
    fout.write('%.1f,%.1f,%.8e\n' % (x,y,t))

fout.close()
```

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

```python
import numpy as np
#import matplotlib
#matplotlib.use('Agg')
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import pyproj

manualFix = True

# http://spatialreference.org/ref/epsg/26713/
# http://spatialreference.org/ref/epsg/32012/
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

year = '2013'
fprefix = 'pest_02/'
mprefix = '../../wipp-polyline-data/'
cfname = fprefix + 'modeled_head_pest_02.grd'
pfname = fprefix + 'dtrk_output_pest_02.bln'
wfname = fprefix + '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
wellutmx,wellutmy = np.loadtxt(wfname,skiprows=1,usecols=(1,2),unpack=True)
wellx,welly = transform(wellutmx,wellutmy)
names = np.loadtxt(wfname,skiprows=1,usecols=(0,),dtype='|S6')

#print 'DEBUG well coordinates'
#for n,ux,uy,x,y in zip(names,wellutmx,wellutmy,wellx,welly):
#    print n,ux,uy,'::',x,y

# 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
```

```python

# 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(mprefix+'total_boundary.dat',unpack=True))
wipputmx,wipputmy = np.loadtxt(mprefix+'wipp_boundary.dat',
                               usecols=(0,1),unpack=True)
wippx,wippy = transform(wipputmx,wipputmy)
aserx,asery = transform(*np.loadtxt(mprefix+'ASER_boundary.csv',
                               delimiter=',',usecols=(1,2),unpack=True))

#print 'DEBUG WIPP coordinates'
for ux,uy,x,y in zip(wipputmx,wipputmy,wippx,wippy):
    print ux,uy,'::',x,y

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='.',
        markeredgecolor='green',markerfacecolor='green')
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_title('Freshwater Heads Model Area '+year)
ax.set_xlabel('NAD27 NM East State Plane Easting (ft)')
ax.set_ylabel('NAD27 NM East State Plane Northing (ft)')

# compute travel time and path length to WIPP LWB
# *****************************************

# compute incremental distance between times
pd = M2FT*np.sqrt((px[1:]-px[:-1])**2 + (py[1:]-py[:-1])**2)

ax.text(688000,537000,'MODFLOW Active Flow Boundary',size=12,rotation=-26,color='purple')
ax.annotate('WIPP LWB',xy=(670000,509200),xytext=(675000,515000),
            fontsize=12,arrowprops=dict(facecolor='black',width=1))
ax.annotate('ASER Contour Area',xy=(658000,478500),fontsize=12,color='green')

print 'particle length:',pd.sum(),' (meters);  travel time:',part[-1],' (years); ',
print '  avg speed:',pd.sum()/part[-1],'(m/yr)'

if manualFix:
    # manually fix labels
    for lab in a:
        lab.draggable()
    plt.show()
else:
    plt.savefig('large-area-contour-map'+year+'.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='.',
        markeredgecolor='green',markerfacecolor='green')
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()])
```

```python
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_title('Freshwater Heads WIPP Area '+ year)
ax.set_xlabel('NAD27 NM East State Plane Easting (ft)')
ax.set_ylabel('NAD27 NM East State Plane Northing (ft)')

ax.annotate('WIPP LWB',xy=(665000,488200),fontsize=12)
ax.text(678700,495000,'MODFLOW No-Flow Area',size=16,rotation=-90,color='purple')

if manualFix:
    # manually fix labels>>>>
    for lab in a:
        lab.draggable()
    plt.show()
else:
    plt.savefig('aser-area-contour-map'+year+'.png')
plt.close(1)
```

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

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

fprefix = 'pest_02/'
mprefix = '../../wipp-polyline-data/'
fname = fprefix + 'modeled_vs_observed_head_pest_02.txt'

ofname = 'original_average/modeled_vs_observed_head_original_average.txt'

M2FT = 0.3048
year = '2013'

# load in observed, modeled, obs-mod, (all in meters)
res = np.loadtxt(fname,skiprows=1,usecols=(3,4,5))
ores = np.loadtxt(ofname,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')

# load in N/S/C/X zones
zones = np.loadtxt('obs_loc_%sASER.dat' % year,usecols=(2,),dtype='|S1')

## checking locations / zones
# ********************************************
wipp = np.loadtxt(mprefix+'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(mprefix+'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
colors = {'N':'red','S':'blue','C':'green','X':'gray'}
ax1.plot(buff[:,0],buff[:,1],'g--') # WIPP LWB+3km
for xv,yv,n,w,z in zip(x,y,names,weights,zones):
    print xv,yv,n,w,z
    plt.annotate('%s %i'%(n,w),xy=(xv,yv),fontsize=8,color=colors[z])
plt.axis('image')
ax1.set_xlim([x.min()-1000,x.max()+1000])
```

```python
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,z in zip(x,y,names,weights,zones):
    plt.annotate('%s %i '%(n,w),xy=(xv,yv),fontsize=8,color=colors[z])
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.suptitle('well weights check '+year)
plt.savefig('check-well-weights-'+year+'.png')

# convert lengths to feet
res /= M2FT
ores /= 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.title('Histogram of Model Residuals '+year)
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-'+year+'.png')
plt.close(1)

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

m0 = weights==0
```

```python
m1 = weights==1
m2 = np.logical_or(weights==2,weights==99)

# separate wells into groups
resin =    res[m0,2]
resnear = res[m1,2]
resfar =   res[m2,2]

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

# separate names into groups
namin =     names[m0]
namnear = names[m1]
namfar =   names[m2]

# 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])

print ab.shape
print ab

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]])
```

```python

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')

ax.set_ylabel('Measured-Modeled (ft)')
ax.set_title('individual residuals '+year)
plt.annotate('AEC-7 @ %.1f'%res[0,2],xy=(nin+nnear+1.0,-14.5),xycoords='data')

plt.savefig('model-error-residuals-'+year+'.png')
plt.close(1)


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

fh = open('calibration-statistics-%s.csv' % year,'w')

fh.write('wellgroup,calibrated,uncalibrated\n')
fh.write('"all wells",%.4f,' % np.corrcoef(res[:,0],res[:,1])[1,0]**2)
fh.write('%.4f\n' % np.corrcoef(ores[:,0],ores[:,1])[1,0]**2)

fh.write('"wells inside 3km of LWB",%.4f,' % np.corrcoef(res[weights<2,0], res[weights<2,1
fh.write('%.4f\n' % np.corrcoef(ores[weights<2,0], ores[weights<2,1])[1,0]**2)

fh.write('"wells ~inside LWB",%.4f,' % np.corrcoef(res[weights==0,0],res[weights==0,1])[1,
fh.write('%.4f\n' % np.corrcoef(ores[weights==0,0],ores[weights==0,1])[1,0]**2)

fh.close()

fig = plt.figure(1,figsize=squarefig)
ax = fig.add_subplot(111)
ax.plot(res[m0,0],res[m0,1],color='red',markersize=10,
        marker='+',linestyle='none',label='Inside LWB')
ax.plot(res[m1,0],res[m1,1],color='green',markersize=10,
        marker='x',linestyle='none',label='< 3km From LWB')
ax.plot(res[m2,0],res[m2,1],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)')
ax.set_title('modeled vs. measured '+year)
plt.savefig('scatter_pest_02_'+year+'.png')
```