Explanatory Notes
for the
Milford GMDSI Worked Example Report

John Doherty

July, 2021

Table of Contents

1. Introduction

1.1 General

This document is a companion to the following GMDSI worked example:

> Doherty, J., Rumbaugh, J. and Muffels, C., 2021. Probabilistic Contributing Area Analysis. GMDSI worked example. ISBN: 978-1-925562-57-6. Downloadable from http://www.gmdsi.org.

This document is not so much a tutorial as it is an explanation. It describes files that are important components of the work that is reported in the worked example. It also describes how they are produced and how they are used. The files themselves are provided with this document.

Different sections of this document address different modelling tasks that are described in the worked example report. These tasks include:

- Model construction and parameterisation;
- Calibration;
- Sampling of the posterior parameter probability distribution;
- Probabilistic contributing area analysis;
- Visualisation and display.

The Milford worked example report describes two models – a three layer model and a single layer model. Only the three layer model is discussed herein.

1.2 Files, Folders and Programs

In different sections of this document, your attention is drawn to different working folders. These folders contain files that are the focus of the current discussion. Many files are common to all of these working folders. Some files have the same name, but have had their contents altered to accommodate the current task. Some new files are introduced to working folders as the workflow progresses.

All of these working folders contain a set of subfolders. The following three subfolders are common to all of them:

- *exec*
- *observ_files*
- *param_files*

The second and third of these subfolders contain files that are important to history-matching. Conceptually, the first of these subfolders contains executable programs that are required for running the model and its pre/post-processors when undertaking history-matching and probabilistic contributing area analysis. The folders supplied with this document include some of these executable programs – namely those that belong to the PEST suite and the PEST Groundwater Utility suite. As time goes on, these executable programs may become outdated as newer versions are produced. So when using PEST-suite programs in your own modelling work, do not use these versions. The latest versions can be downloaded from the following site:

> https://www.pesthomepage.org

Some programs which were included in *exec* subfolders when we developed and deployed the Milford model are not included in versions of this subfolder that are supplied with this document. These are programs that do not belong to the PEST suite. (These are not mine to circulate.) Hence if you try to reproduce any of the workflow steps described herein, you will need to place the following files in various the *exec* subfolders that are supplied with this document:

- *mp3du.exe* and *writep3doutput.exe*; these are supplied with mod-PATH3DU.
- *pestpp-ies.exe* and *pestpp-swp.exe*; these are part of the PEST++ suite.
- *USGs_1.exe*; this is the MODFLOW-USG executable program.

Mod_PATH3DU can be downloaded from:

https://www.sspa.com/software/mod-path3du

Programs of the PEST++ suite can be downloaded from:

https://github.com/usgs/pestpp/tree/master

MODFLOW-USG can be downloaded from:

https://www.gsi-net.com/en/software/free-software/modflow-usg.html

The Milford model must be run from the working folder (i.e. the parent folder of the *exec* subfolder). So too must PEST_HP and its agent AGENT_HP. The same applies to PESTPP-IES and PESTPP-SWP. Model batch files that are run by these programs affix a relative path to the *exec* subfolder when they run MODFLOW-USG or a model pre/postprocessor program. When parallelizing model runs, a copy of the working folder and all of its subfolders must be copied to locations from which parallelisation agents operate.

A number of PEST-support programs that are not actually required for running the Milford model, but are used in file preparation, or in visualisation and display, are discussed in this document. These programs are not included in *exec* subfolders. Download them from the PEST website if you wish to use them.

2. The MODFLOW-USG Model

2.1 General

Our workflow commences with a set of model input files that were produced by Groundwater Vistas (GV). A step-by-step guide to construction of the Milford model is provided in a GV tutorial. However, as is now described, some of the files produced by GV require modification so that the model is more easily amenable to PEST/PEST++ history-matching.

As is described in the Milford worked example report, the simulator is MODFLOW-USG (MFUSG). Even though the Milford model grid is actually structured, MFUSG files characterise it as unstructured. Hence model cells are identified by node number rather than by row, column and layer numbers. The explanations and instructions that are described in the present document are thus applicable to all unstructured grid MFUSG models.

Files that are described in the present section reside in a folder named *calibration*.

2.2 MFUSG Input Files

2.2.1 Name File

The name file for the Milford model is *milford1.nam*. This is situated in the working folder (i.e. the *calibration* folder). The "1" suffix was inherited from an earlier stage of model construction. This is a little messy. I could have enhanced the aesthetic appeal of this name by removing the "1". However I hesitate to alter the name of an important model input file, as this has the potential to create a ripple effect of errors.

2.2.2 MFUSG Package Input Files

As is readily apparent from an inspection of *milford1.nam*, the filename base of most other model input files is also *milford1*.

2.2.3 External Files

Two packages direct MFUSG to read model node (i.e. cell) data from external files.

Examine *milford1.rch* – the input file for the RCH (i.e. recharge) package. A recharge array is required for only one stress period because the model is run under steady state conditions. The "external" statement on the fourth line of this file directs MFUSG to read this array from a file that is connected to unit 33. An inspection of the name file reveals that the pertinent file is *recharge.dat*. Inspection of *recharge.dat* reveals that cell-based recharge values are listed one to a line. This is not essential, as MFUSG allows more than one number to be recorded on each line of an external file; however, if the latter protocol is followed, numbers for each new layer must begin on a new line. Of course, the "layer" concept does not apply for recharge. Nevertheless, I personally find the listing of cell-based model inputs one to a line convenient as an external MFUSG input file that adopts this protocol can then be easily converted to a node data table file; this type of file is described below.

Inspect file *milford1.lpf* – the input file for the LPF (i.e. layer property flow) package. The "external" protocol is used for horizontal and vertical hydraulic conductivities (i.e. kx and kz). All kx values for all model cells are read from unit number 31, while all kz values are read from unit number 32. An inspection of *milford1.nam* reveals that these unit numbers are connected to files *kx.dat* and *kz.dat*. With one kx or kz value per line, these files therefore contain as many lines as there are nodes in the model grid (190080 in the present case). As is explained in documentation of the PEST Groundwater Utilities, conversion of this file to a node data table file only requires:

- The addition of a leading column containing the numbers 1 to 190080; and

- The provision of a header for each column; the header for the first column must be "node" or "node_number".

By the way, if you ever need to know the number of nodes in a MFUSG unstructured grid, this is easily ascertained. It is the first number on the first line of the DIS (i.e. discretisation) file; for the Milford model this is *milford1.dis*.

External files such as *recharge.dat*, *kx.dat* and *kz.dat* are re-written on every occasion that a model is run by PEST. The task of supplying a model with a new set of hydraulic properties is particularly easy when these properties are recorded in external files.

2.3 MFUSG Output Files

2.3.1 Heads

MFUSG is instructed to record the heads that it calculates in a binary file named *milford1.hds*. Heads in observation wells form part of the calibration dataset.

2.3.2 Water Budgets

As is described in the worked example report, other components of the calibration dataset include:

- exchange of water with RIV boundaries;
- exchange of water with WEL boundaries;
- exchange of water with DRN boundaries.

Cell-by-cell flows are recorded in file *milford1.cbb*. However, when the model is run for calibration purposes, MFUSG is instructed to record inter-cell flows only for the above packages. This reduces the size of *milford1.cbb*; it also reduces MFUSG's burden in writing/re-writing this file. This is accomplished by setting the IRIVCB, IWELCB and IDRNCB variables in respective MFUSG package input files to 50, the unit number that is connected to *milford1.cbb*. In contrast, the ILPFCB variable in file *milford1.lpf* (the input file for the LPF package) is set to 0 in order to disable cell-by-cell flow term output for this package. (Note that it is important to re-instate cell-by-cell flow term output prior to calculation of particle tracks, for particle tracking requires knowledge of all inter-cell flows.)

The WEL package is used for two purposes in the Milford model. Lateral recharge into layer 3 of the model is simulated using this package. The WEL package is also employed to simulate withdrawal of water from production wells. Water is withdrawn from CLN elements.

2.3.3 Automatic Flow Reduction

The first line of the WEL package input file *milford1.wel* includes the text string "AUTOFLOWREDUCE". This instructs MFUSG to reduce WEL extraction (from CLN nodes) if the head approaches the bottom of the element from which water is extracted. This forestalls calculation of excessive local drawdowns and reduces the chances of numerical instability. If MFUSG needs to reduce extraction from any CLN element, it records the adjusted extraction rate in a file whose unit number follows the IUNITAFR keyword; this is unit number 166. A perusal of the name file reveals that the file to which this unit number is connected is *milford1_FlowReduction.dat*.

As is discussed in the Milford worked example report, the calibration dataset includes observations which discourage the inversion process from introducing parameter sets which precipitate flow reduction. Unfortunately, however, the information recorded in file *milford1_FlowReduction.dat* cannot be read by PEST in order to monitor flow reduction because its contents depend on whether or not flow reduction is actually required; it also depends on the number of wells for which flow reduction is implemented. Design of a PEST instruction set to read a model output file requires that the contents of that file do not change dramatically from model run to model run.

An alternative strategy is therefore employed to detect and mitigate reduction of WEL extraction. Water withdrawals from extraction wells are recorded as cell-by-cell flow terms in a MFUSG budget (i.e. cell-by-cell flow term) file. Where wells extract water from CLN nodes rather than GWF nodes, these flow terms pertain to CLN nodes rather than to GWF nodes. The recording of CLN cell-by-cell flow terms is activated by setting the ICLNCB variable to an appropriate unit number in the MFUSG CLN package input file *milford1.cln*. In the present case, it is set to 144. An inspection of *milford1.nam* reveals that this unit number is connected to file *milford1.cbcln*.

2.4 An Extra MFUSG File

File *.\param_files\milford1.gsf* is a "grid specification file" for the Milford model. This is not a standard MFUSG file. However it is required by any model postprocessing program which needs to know the real-world coordinates of model grid nodes, and/or model grid cell vertices. These programs include:

- utility programs which implement spatial interpolation to or from the model grid;
- programs which enable visualisation and display of the model and its properties/results;
- particle tracking programs such as mod-PATH3DU which guide particles through model cells.

The grid specification file was written by GV.

2.5 Preparing for PLPROC

2.5.1 What is PLPROC?

Pilot point parameterisation and calibration of the Milford model is described in the next section of this document. Numerical tasks required for use of these (and other) parameterisation devices are performed by PLPROC. "PLPROC" stands for "parameter list processor". It can be downloaded from the PEST web pages, along with a comprehensive manual.

2.5.2 Model Information Node Data Table File

2.5.2.1 General

As will be described in the next section, history-matching of the Milford model requires that PEST estimate certain hydraulic properties (for example horizontal and vertical hydraulic conductivities), and that it modifies other hydraulic properties (such as riverbed conductance and lateral recharge) that were provided by GV when it built the model. These latter model inputs must be recorded in a format that PLPROC can read.

Inspect file *.\param_files\model_info.ndt*. This is a "node data table file". This type of file is employed extensively by those members of the PEST Groundwater Utility suite which manipulate MFUSG datasets. The protocol is simple.

- The first column must list model node numbers sequentially; none must be omitted.
- Subsequent columns list integers or real numbers associated with each node.
- Each column has a header (of 20 characters or less in length).
- The header for the first column must be "node" or "node_number".

This file must be prepared by the user. Some columns are easy to prepare (for example the first). Others require a bit of software assistance. Use of a text editing program that supports copying and pasting of columns is essential.

Preparation of each of the data columns appearing in file *model_info.ndt* is now discussed.

2.5.2.2 IBOUND

Model activity arrays are recorded in file *milford1.bas*. These data (as well as other data) can be extracted from a MFUSG model input file using the USGPROP2TAB1 or USGPROP2TAB2 utilities belonging to the PEST Groundwater Utility suite. To extract cell IBOUND values and record them in a node data table file named *temp.ndt*, run USGPROP2TAB1. Respond to its prompts as follows:

```
Enter name of MODFLOW-USG model input file: milford1.bas

Enter text string for array recognition in this file: ibound
Are these arrays integer or real? [i/r]: i

Enter total number of nodes in grid/network: 190080

Enter name for node data table file: temp.ndt
Enter value to assign to uncited nodes: 0
```

2.5.2.3 Layer

Numbers comprising the "layer" column of file *model_info.ndt* can be extracted from the second part of the grid specification file *.\param_files\milford1.gsf* using the column cut and paste functionality of a text editor. (Yes, this is a little painful). Layer numbers comprise the fifth column in the second (and final) part of this file.

2.5.2.4 Riverbed Conductance and River Head

These occupy the fourth and fifth columns of file *model_info.ndt*.

Conductances and heads are assigned to RIV nodes in the MFUSG input file *milford1.riv*. As there is only one stress period, only one table of RIV properties is recorded in this file. However this table does not comply with node data table format as it does not cite every cell in the model grid.

Program TAB2NDTF from the PEST Groundwater Utility suite can reformat this table. First, copy *milford1.riv* to a separate file, say *temp.dat*. Then remove the top three lines from this file so that only the table remains. Respond to TAB2NDTF's prompts as follows in order to record some of the data that resides in this table in a node data table file named *temp.ndt*.

```
Enter name of tabular data file: temp.dat

Skip how many lines at the top of this file? 0
Enter column containing node numbers: 1

Enter column number from which to read node data (<Enter> if no more): 3
Enter a label for this data (12 characters of less): riv_cond
Is this integer or real data?  [i/r]: r
Enter data value to assign to uncited nodes: 0

Enter column number from which to read node data (<Enter> if no more): 2
Enter a label for this data (12 characters of less): riv_head
Is this integer or real data?  [i/r]: r
Enter data value to assign to uncited nodes: 0

Enter column number from which to read node data (<Enter> if no more): <Enter>

Enter name for node data table file: temp.ndt

Enter total number of nodes in model grid: 190080
```

Columns from *temp.ndt* can be added to columns in an existing node data table file using the column cut and paste functionality of a text editor.

2.5.2.5 River Reach

In construction of the Milford model, streams are divided into two types of reach. One set of reaches is used for parameterisation purposes, while the other set of reaches is used for observations. In the

Milford worked example report, the former set of reaches are referred to as "segments" to save confusion. However we retain the word "reach" in the present document in order to avoid altering some of the files that are discussed herein.

"Parameter reaches", or segments, are used for pilot point interpolation. Each pilot point belonging to the *pp_riv* family of pilot points that is used for RIV parameterisation (see the next section) is assigned to a river segment. Each river cell is assigned to a river segment. Interpolation from a pilot point to a RIV cell can occur only if both belong to the same segment. As is explained in the worked example report, segment-based interpolation is useful when assigning values to cells that comprise polylinear features such as rivers. It prevents pilot points in one segment from affecting model cells in another segment, even where these different segments are close together (as happens when a tributary joins a river). River segments and associated pilot points share common colours in Figure 2.1.

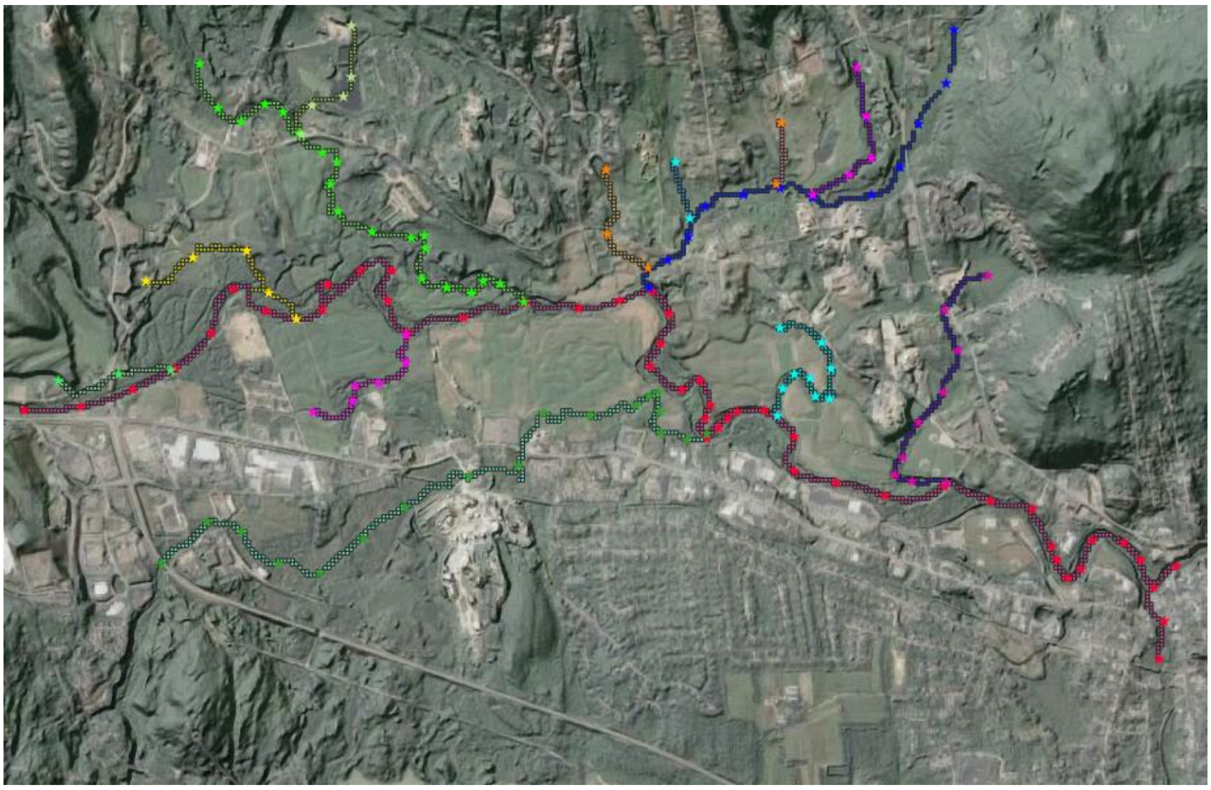

Figure 2.1. Model cells to which RIV boundary conditions are introduced. These are coloured by parameter reach (i.e. segment). Pilot points which interpolate to these segments are also shown.

River cells were assigned segment numbers in the QGIS package. These assignments were exported as a table. The table was reformatted as a node data table file using TAB2NDTF. Column cutting and pasting was then used to add the river reach column to *model_info.ndt*.

Transfer of data between MFUSG input and output files and a GIS platform makes use of the USGNDTF2MIF program supplied with the Groundwater Utilities. See Section 8 of this document.

2.5.2.6 Well Rate

A node data table file of WEL injection rates can be prepared from the *milford1.wel* MFUSG input file using TAB2NDTF in the same manner as was described above for RIV package data. As is discussed above, lateral recharge to the Milford model is introduced into layer 3 of that model using the WEL package. It is important to note that the last 6 lines must be removed from the table provided in *milford1.wel* before running TAB2NDTF, as these lines pertain to production wells and not to lateral recharge. Furthermore, node numbers that are cited in these lines pertain to CLN elements rather than to GWF elements.

2.5.2.7 Well_N_or_S

In this column of file *model_info.ndt*, an integer value of 1 is assigned to any cell which contains a WEL boundary condition if this cell is on the northern boundary of the Milford model; a value of 2 is assigned to any cell that contains a WEL boundary condition if this cell is on the southern boundary of the model. Values of 0 are assigned to all other cells. These values were assigned in QGIS. The table of values was then exported. TAB2NDTF was used to create a node data table file. The column of integers was then placed into *model_info.ndt* using the column cut-and-paste functionality of a text editor.

3. Model Parameterisation

3.1 Overview

Parameterisation of the Milford model is based entirely on pilot points. Reading of pilot point coordinates and associated parameter values, interpolation of these values to the model grid, and manipulation of model hydraulic property arrays prior to writing them to model input files is carried out by the PLPROC parameter pre-processor.

The PEST control file on which calibration of the Milford model is based is *milford1.pst*. Parameters are listed in the "parameter data" section of this file. Among other things, this section assigns initial values to parameters, as well as upper and lower bounds to parameters. As is discussed below, initial parameter values are also "preferred" parameter values; preferred values are applied through Tikhonov regularisation.

Parameter groups are listed in the "parameter groups" section of file *milford1.pst*.

3.2 Families of Pilot Points

For the Milford model most, but not all, of the parameters that are associated with pilot points are multipliers. As such, their initial/preferred values are 1.0. Parameters are named according to the following protocol:

ppset_functionN

where:

ppset	is a pilot point family prefix;
function	describes the role of the parameter; and
N	is the parameter index.

Parameter indices start at 1 for each pilot point family, and are sequential.

Table 3.1 lists the roles of different families of pilot points. Figures which show pilot point locations are provided after that.

Pilot point family	Prefix	Function	Description
ppset1	pp1	kx	“Primary” (or “base”) hydraulic conductivity in layer 2.
ppset2	pp2	mul	Multiplies primary layer 2 hydraulic conductivity field to obtain final layer 2 hydraulic conductivity field, kx2.
ppset3	pp3	mul	Multiplies kx2 to obtain kx for layer 1 (i.e. kx1); divides kx2 to obtain kx for layer 3 (i.e. kx3).
ppset3	pp3	vanis	Divides kx1, kx2 and kx3 to obtain kz1, kz2 and kz3 (i.e. vertical hydraulic conductivities).
ppset3	pp3	rech	Recharge.
pp_riv	ppr	mul	Multiplies river conductances provided by GV.
pp_riv	ppr	add	Adds to river elevations provided by GV.
pp_wel	ppw	mul	Multiplies well inflow rates along northern/southern model boundaries provided by GV.

Table 3.1. Pilot point families used in parameterisation of the Milford model.

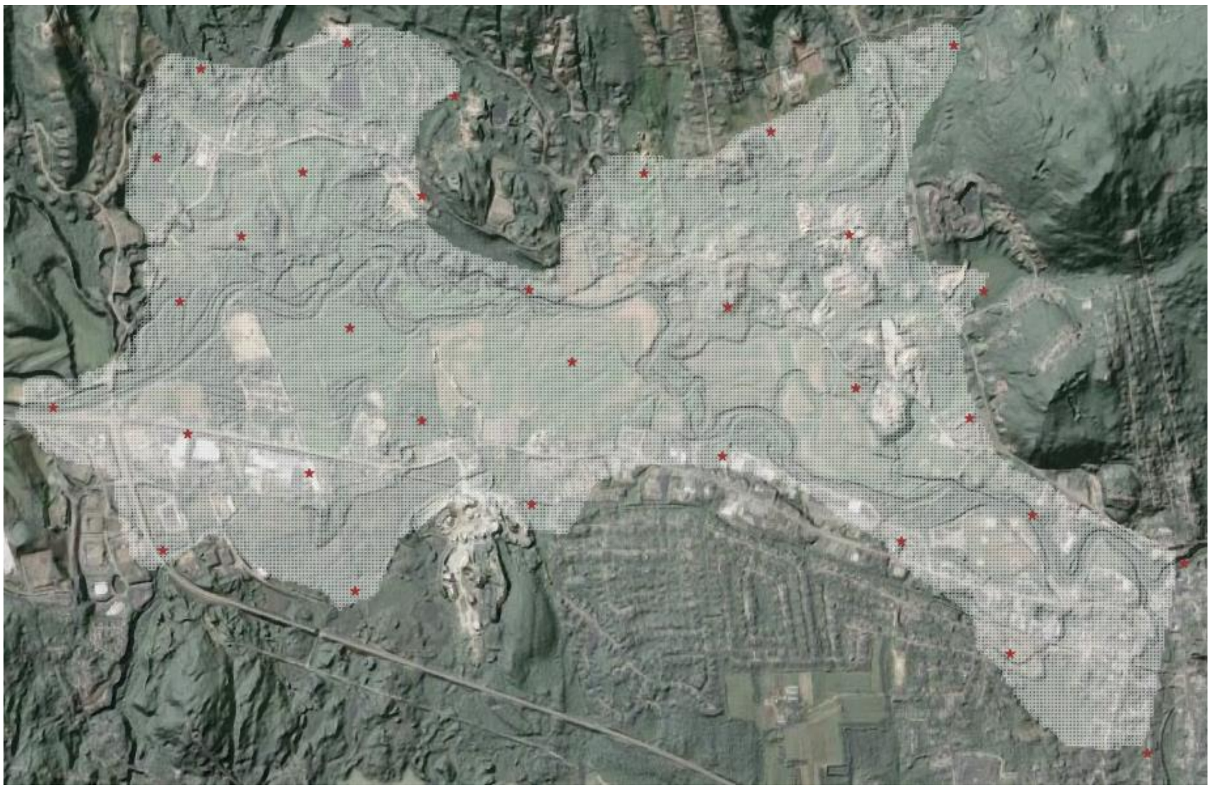

Figure 3.1a. Pilot points belonging to *ppset1*.

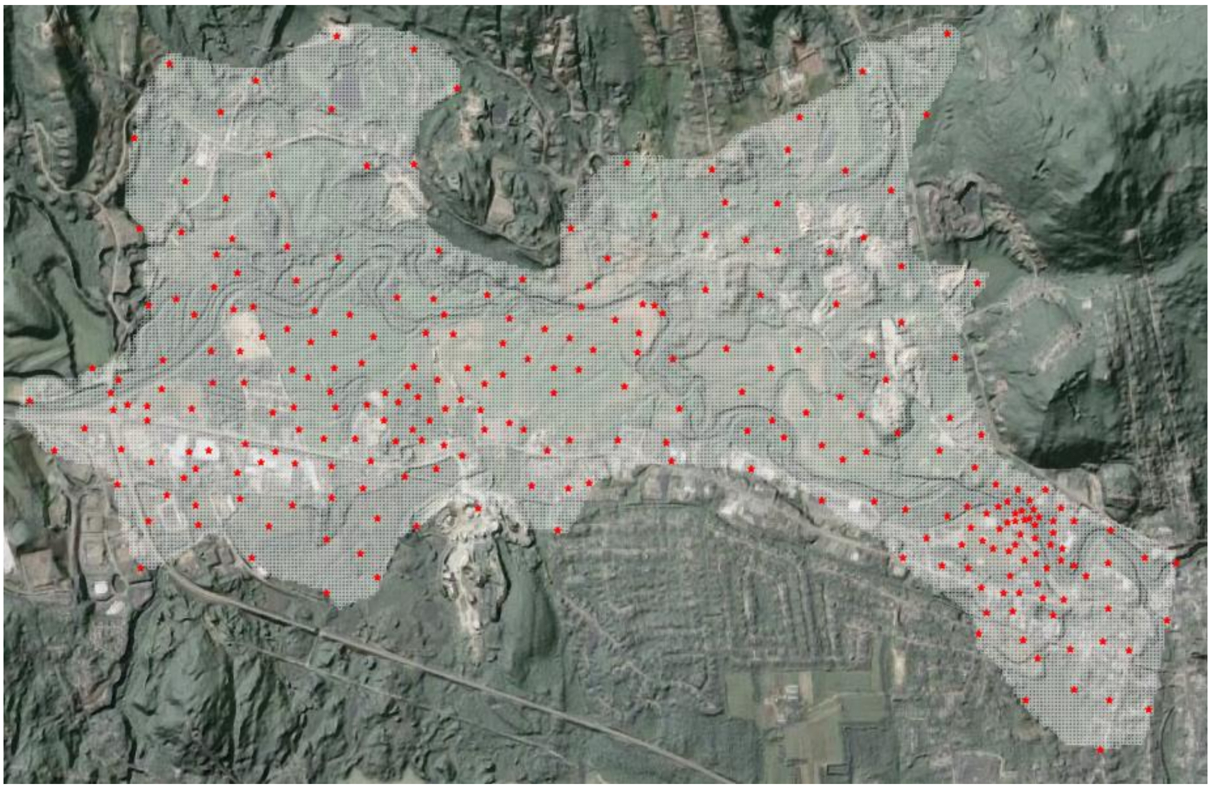
Figure 3.1b. Pilot points belonging to *ppset2*.

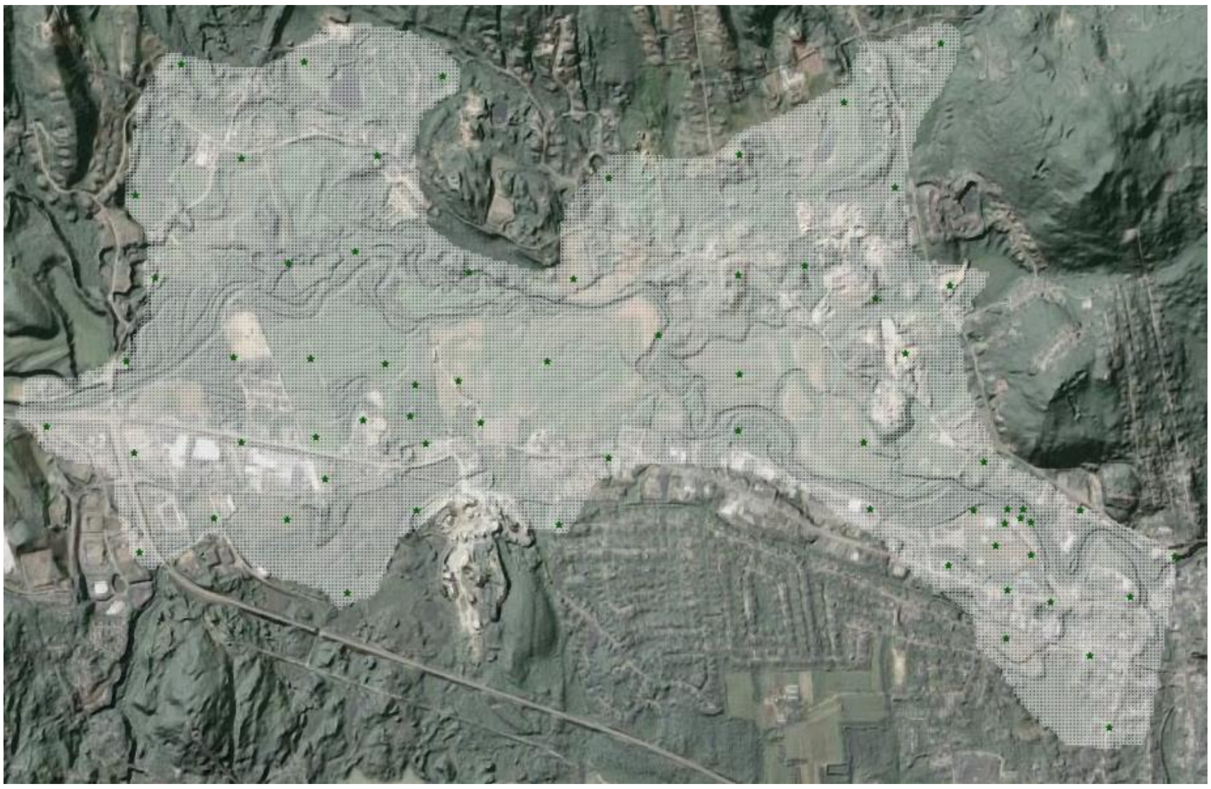
Figure 3.1c. Pilot points belonging to *ppset3*.

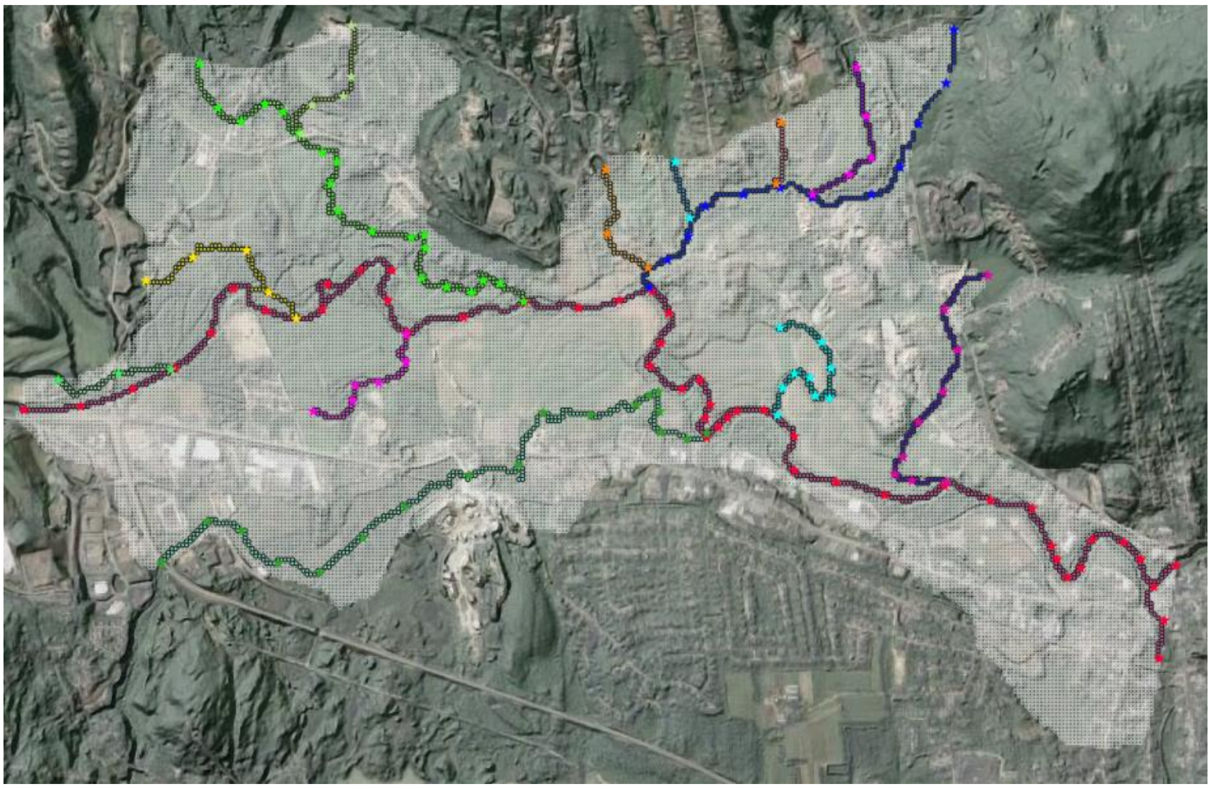

Figure 3.1d. Pilot points belonging to *pp_riv*. Also shown are the cells to which they interpolate; these are all in model layer 1.

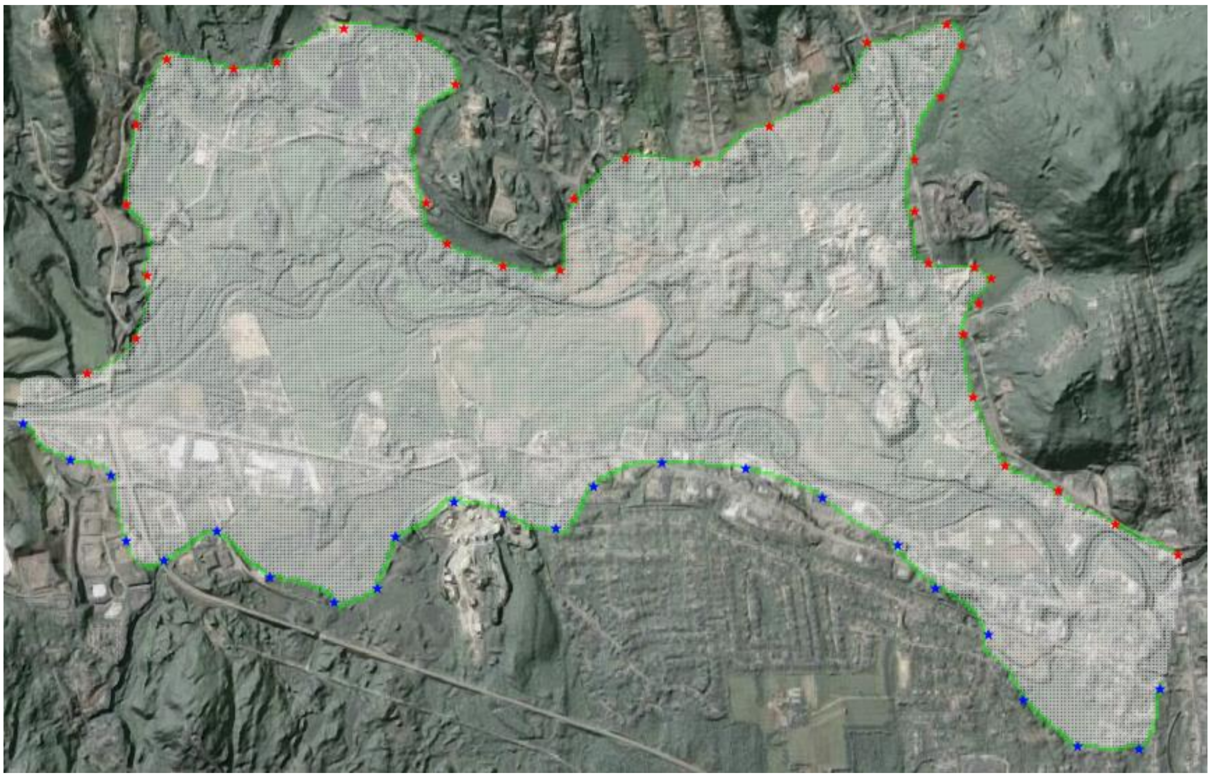

Figure 3.1e. Pilot points belonging to *pp_wel*. Also show are the cells to which they interpolate; these are all in model layer 3.

3.3 PLPROC

3.3.1 Running PLPROC

Whenever PEST runs the Milford model, it runs a model batch file named *model.bat*. Included in this file is the command to run MFUSG. Preceding the command to run MFUSG is the command to run PLPROC. The PLPROC executable is supplied in the *.\exec* folder. *model.bat* provides the following command to run it.

```
.\exec\plproc plproc.dat
```

plproc.dat is the PLPROC input file.

The script that is provided in *plproc.dat* is now explained. For descriptions of entities such as CLISTs, PLISTs and SLISTs, refer to documentation of PLPROC. The PLPROC manual also describes all of the functions that PLPROC supports.

As has already been discussed, even though the Milford model employs a structured grid, MFUSG uses unstructured grid protocols to refer to model cells. So too does PLPROC. Hence all aspects of the PLPROC script that are demonstrated below are applicable to any unstructured grid MFUSG model.

3.3.2 Reading Model Data

3.3.2.1 Model Structure

PLPROC's first function call is:

```
cl_mod = read_mf_usg_grid_specs(file=.\param_files\milford1.gsf,slist=layer)
```

This function tells PLPROC to read the MFUSG grid specification file. Once it has done this, it creates a CLIST named *cl_mod*. This is a three-dimensional CLIST. Each element of this CLIST corresponds to a node (i.e. cell) of the MFUSG model grid. PLPROC also creates an SLIST that is dependent on this CLIST. The SLIST is named *layer*. The integer associated with each model node in this SLIST is the layer to which the node belongs.

In processing that follows, PLPROC works with PLISTs and SLISTs that pertain to the whole Milford model grid. It also works with PLISTs and SLISTs that pertain to just one layer of this grid. As all layers have the same number of cells (and the same cell IBOUND values), layer 1 is used as a proxy for all other layers when undertaking operations that are two-dimensional in nature (such as interpolation from pilot points to the model grid).

The following command creates a two-dimensional CLIST named *cl_lay* that pertains to only layer 1 of the Milford model.

```
cl_lay=cl_mod.partition_by_eqn(select=(layer==1))
```

We now extract the IBOUND array for layer 1 from the three-dimensional IBOUND array for the entire Milford model to create a two-dimensional IBOUND array to complement the *cl_lay* CLIST.

```
ibound_lay=ibound.partition_by_clist(clist=cl_lay)
```

3.3.2.2 Model Data

Now data that was previously placed into file *.\param_files\model_info.ndt* is read. As is described in the previous section of this document, this is a node data table file that lists cell-based data, some of which was provided to the Milford model when it was created by GV. As PLPROC reads this file, the contents of its columns are transferred to various three-dimensional PLISTs and SLISTs, depending on whether a particular column hosts real numbers of integers. All of the newly-created PLISTs and SLISTs have *cl_mod* as their parent CLIST.

```
read_list_file(reference_clist='cl_mod',skiplines=1,      &
                        slist='ibound';column=2,          &
                        plist='riv_cond';column=4,        &
                        plist='riv_head';column=5,        &
                        slist='riv_reach';column=6,       &
                        plist='well_rate';column=7,       &
                        slist='well_bnd';column=8,        &
                        file='.\param_files\model_info.ndt')
```

3.3.3 Reading Pilot Point Data

PLPROC now reads a set of files which contain parameter values that are associated with pilot points. These files have certain things in common.

- They all reside in the *.\param_files* subfolder.
- Their initial line contains column headers (which PLPROC ignores).
- Their first column contains pilot point identifiers. To make things easy, these are sequential integers starting at 1.
- Their second and third columns contain pilot point *x* and *y* coordinates. A column for z coordinates is not needed because all of these pilot point families are two-dimensional. (PLPROC is informed of this in each function call).
- One or a number of their ensuing columns contain values that PLPROC must read.

On reading *x* and *y* coordinates from these files, PLPROC defines a series of pilot-point-specific CLISTs. It then populates PLISTs based on data that it reads from other columns of these files. Because these data are actually parameter values, PEST template files must complement all of these pilot point files. These also reside in the *.\param_files* subfolder. Each is named by affixing a "*.tpl*" extension to the respective pilot point data file.

First, horizontal conductivities associated with the *ppset1* family of pilot points are read.

```
cl_pp1 = read_list_file(skiplines=1,dimensions=2,         &
                        plist=pp1_kx;column=4,            &
                        id_type=integer,                  &
                        file=.\param_files\ppset1.dat)
```

Next, multipliers associated with *ppset2* pilot points are read.

```
cl_pp2 = read_list_file(skiplines=1,dimensions=2,         &
                        plist=pp2_kxmul;column=4,         &
                        id_type=integer,                  &
                        file=.\param_files\ppset2.dat)
```

Values of "K-layer-gradient" (see below), vertical anisotropy and recharge that are associated with the *ppset3* family of pilot points are next read.

```
cl_pp3 = read_list_file(skiplines=1,dimensions=2,         &
                        plist=pp3_kxlaymul;column=4,      &
                        plist=pp3_vanis;column=5,         &
                        plist=pp3_recharge;column=6,      &
                        id_type=integer,                  &
                        file=.\param_files\ppset3.dat)
```

A file named *.\param_files\pp_wells.dat* is read next. This pertains to the *pp_wel* family of pilot points. The fourth column of this file contains integers; these denote whether each pilot point belongs to the northern (integer value of 1) or southern (integer value of 2) boundary of the model domain. Meanwhile the fifth column contains parameter values; as is discussed above, these are multipliers on GV-assigned recharge inflows to these boundaries.

```
cl_ppw = read_list_file(skiplines=1,dimensions=2,          &
                        slist=ppw_bnd;column=4,            &
                        plist=ppw_mul;column=5,            &
                        id_type=integer,                   &
                        file=.\param_files\pp_wells.dat)
```

Finally, parameter values associated with the *pp_riv* family of pilot points are read. Integers which denote river segment numbers are read from the fourth column; being integers, they are assigned to an SLIST. River conductance multipliers are read from the fifth column, while river elevation adjustments are read from the sixth column.

```
cl_ppr = read_list_file(skiplines=1,dimensions=2,          &
                        slist='ppr_reach';column=4,        &
                        plist='ppr_mul';column=5,          &
                        plist='ppr_add';column=6,          &
                        id_type='integer',                 &
                        file=.\param_files\pp_rivers.dat)
```

3.3.4 Calculation of Hydraulic Conductivity

Interpolation from pilot points to the Milford model grid is implemented using kriging for three of the five families of pilot points that are used to parameterize the Milford model. These families all host parameters which control the assignment of hydraulic conductivity to cells of this model. Recall from the Milford worked example report that steps required for construction of model hydraulic conductivity arrays are as follows.

1. Cells in layer 2 of the Milford model are assigned hydraulic conductivities through direct interpolation from the *ppset1* family of pilot points.
2. The interpolated hydraulic conductivity array for this layer is multiplied by a model-based array that is interpolated from the *ppset2* family of pilot points.
3. A “K-layer-gradient” array is interpolated to every cell in model layer 2 from a set of parameters that are associated with the *ppset3* family of pilot points.
4. Every cell in layer 1 is assigned a value of hydraulic conductivity by multiplying the hydraulic conductivity of the underlying cell by the interpolated K-layer-gradient value.
5. Every cell in layer 3 is assigned a value of hydraulic conductivity by dividing the hydraulic conductivity assigned to the overlying cell by the interpolated K-layer-gradient value.
6. Vertical hydraulic conductivity anisotropy is interpolated to a representative layer of the model from the *ppset3* pilot point family.
7. The vertical hydraulic conductivity of all cells in all layers is obtained by dividing the horizontal hydraulic conductivity by the interpolated value of vertical anisotropy.

All of the above operations are made easier by the fact that the number and disposition of cells is the same in all model layers. It is made even easier because of the fact that IBOUND values are the same in all model layers. Many PLPROC operations can therefore be applied to a single, “representative” layer of the model using PLISTs and SLISTs for which *cl_lay* is the parent CLIST. Operations would be slightly more complex if IBOUND were not the same in all layers. In this case, an easy approach would be to assign a pseudo-IBOUND array to the “representative” model layer; any element of this pseudo-IBOUND array would be non-zero if the corresponding value of any layer-specific IBOUND array is non-zero.

Explanation of the PLPROC script now continues.

3.3.5 Calculating Interpolation Factors

Interpolation factors from the *ppset1*, *ppset2* and *ppset3* families of pilot points to the model grid are calculated using the following three function calls. These factors are stored in files *factors_pp1.dat*, *factors_pp2.dat* and *factors_pp3.dat*; all of these files are placed in the *.\param_files* subfolder. Note that these factors only need to be computed once. They do not need to be re-computed on each occasion that PEST runs the model. These lines can therefore be commented out of the PLPROC script once interpolation factors have been calculated.

```
calc_kriging_factors_auto_2d(target_clist=cl_lay;select=(ibound_lay.ne.0),      &
                             source_clist=cl_pp1,                               &
                             file=.\param_files\factors_pp1.dat)

calc_kriging_factors_auto_2d(target_clist=cl_lay;select=(ibound_lay.ne.0),      &
                             source_clist=cl_pp2,                               &
                             file=.\param_files\factors_pp2.dat)

calc_kriging_factors_auto_2d(target_clist=cl_lay;select=(ibound_lay.ne.0),      &
                             source_clist=cl_pp3,                               &
                             file=.\param_files\factors_pp3.dat)
```

3.3.6 Calculating Hydraulic Conductivity

Two-dimensional PLISTs are now defined to host horizontal and vertical hydraulic conductivity arrays pertaining to layers 1, 2 and 3 of the Milford model. These are based on the *cl_lay* two-dimensional CLIST which is applicable to all model layers. A *work* PLIST is also initialized. The initial values assigned to all elements of these PLISTs prevail if values are not assigned to these elements in later processing. (So-called “selection equations” used in later processing ensure that PLIST values are altered only for elements for which IBOUND is non-zero.)

```
kx1=new_plist(reference_clist=cl_lay,value=100.0)
kx2=new_plist(reference_clist=cl_lay,value=100.0)
kx3=new_plist(reference_clist=cl_lay,value=100.0)
kz1=new_plist(reference_clist=cl_lay,value=10.0)
kz2=new_plist(reference_clist=cl_lay,value=10.0)
kz3=new_plist(reference_clist=cl_lay,value=10.0)
work=new_plist(reference_clist=cl_lay,value=1.0)
```

Values of horizontal hydraulic conductivity assigned to the *ppset1* family of pilot points are now spatially interpolated to cells of the model grid. These interpolated values are assigned to the *kx2* PLIST.

```
kx2=pp1_kx.krige_using_file(file='.\param_files\factors_pp1.dat',transform=log)
```

Hydraulic conductivity multipliers hosted by the *ppset2* family of pilot points are now interpolated and applied.

```
work=pp2_kxmul.krige_using_file(file='.\param_files\factors_pp2.dat',transform=log)
kx2=kx2*work
```

Hydraulic conductivities are assigned to model layers 1 and 3 after interpolation of K-layer-gradient parameters from the *ppset3* family of pilot points to the model grid.

```
work=pp3_kxlaymul.krige_using_file(file='.\param_files\factors_pp3.dat',transform=log)
kx1 = kx2*work
kx3 = kx2/work
```

After interpolating vertical anisotropy parameters associated with the *ppset3* family of pilot points to the model grid, vertical hydraulic conductivities are calculated for cells in the three model layers.

```
work=pp3_vanis.krige_using_file(file='.\param_files\factors_pp3.dat',transform=log)
kz1 = kx1/work
kz2 = kx2/work
kz3 = kx3/work
```

3.3.7 Recording Hydraulic Conductivities on Model Input Files

Recall that MFUSG reads horizontal and vertical hydraulic conductivities from files *kx.dat* and *kz.dat*. PLPROC must therefore populate these files. Like PEST, PLPROC can write numbers on model input files using templates of those files. However the template files that PLPROC uses can be much more complicated than those used by PEST. Whole arrays can be transferred from PLPROC's memory to a model input file using a PLPROC function that is embedded in a template of that file.

PLPROC uses a template file named *kx.dat.tpl* to write *kx.dat*. The contents of *kx.dat.tpl* are as follows.

```
$#p  kx1.write_in_sequence(format="free",number_per_line=1)
$#p  kx2.write_in_sequence(format="free",number_per_line=1)
$#p  kx3.write_in_sequence(format="free",number_per_line=1)
```

An embedded PLPROC function begins with the character string "$#p". The first of the above three functions instructs PLPROC to record elements of the *kx1* PLIST one to a line. The second and third functions do the same for the *kx2* and *kx3* PLISTs. The entire model *kx* dataset is therefore transferred.

In the PLPROC script itself (provided in file *plproc.dat*), PLPROC is directed to use these template files to write respective model input files through the following function calls.

```
write_model_input_file(template_file=.\param_files\kx.dat.tpl,          &
                       model_input_file=kx.dat)

write_model_input_file(template_file=.\param_files\kz.dat.tpl,          &
                       model_input_file=kz.dat)
```

3.3.8 Recharge

Handling of recharge is much less complicated than handling of hydraulic conductivity. First the values of pilot-point-based recharge parameters are interpolated to the model grid using interpolation factors calculated for the *ppset3* family of pilot points (see above). Then cell-based recharge values are written to model input file *recharge.dat* using an embedded PLPROC function recorded in a template of this file named *recharge.dat.tpl*. PLPROC function calls are as follows.

```
recharge=new_plist(reference_clist=cl_lay,value=0.0)
recharge=pp3_recharge.krige_using_file(file='.\param_files\factors_pp3.dat',  &
                                       transform=none)
write_model_input_file(template_file=.\param_files\recharge.dat.tpl,         &
                       model_input_file=recharge.dat)
```

3.3.9 Lateral Recharge

All further interpolation from pilot points to the model grid employs the inverse power of squared distance methodology. Hence there is no need for pre-calculation of kriging factors.

From now on, we work with the *cl_mod* CLIST which pertains to the whole model grid. This is because, as will be demonstrated below, when writing interpolated values of lateral recharge and river properties to the model grid, the linkage between model cells and CLIST elements must be clear; we can no longer use a single representative model layer.

First we define a PLIST based on *cl_mod* which will receive values interpolated from pilot points. A value of 1.0 is assigned to all of its elements (and thereby to all cells of the model grid.)

```
work3d=new_plist(reference_clist=cl_mod,value=1.0)
```

Spatial interpolation of recharge multipliers to cells along the northern boundary of the model domain that receive lateral recharge is now undertaken. Recall that the *well_bnd* SLIST (read from file *model_info.ndt*) identifies these cells. Interpolation takes place from the *pp_wel* family of pilot points, but only from pilot points that are assigned to the northern boundary. The *well_bnd* SLIST identifies whether a model cell belongs to the northern or southern model boundary, while the *ppw_bnd* SLIST identifies whether a pilot point belongs to the northern or southern model boundary.

```
work3d(select=(well_bnd.eq.1))=ppw_mul.ivd_interpolate_2d(       &
                               select=(ppw_bnd==1),              &
                               transform='log',                  &
                               inv_power=2.0,                    &
                               min_points=2,                     &
                               max_points=100,                   &
                               search_radius=20000.0)
```

Interpolation to the southern inflow boundary takes place in similar fashion.

```
work3d(select=(well_bnd.eq.2))=ppw_mul.ivd_interpolate_2d(       &
                               select=(ppw_bnd==2),              &
                               transform='log',                  &
                               inv_power=2.0,                    &
                               min_points=2,                     &
                               max_points=100,                   &
                               search_radius=20000.0)
```

Modification of cell-based inflow rates that were calculated by GV (and read from file *model_info.ndt*) is now carried out.

```
well_rate=well_rate*work3d
```

Next PLPROC must record processed lateral inflow rates on the appropriate model input file. This is a little more complicated than for hydraulic conductivities as MFUSG cannot read WEL data from an external file. Furthermore, only cells which actually receive lateral inflow should be represented in this input file. So PLPROC must modify an existing MFUSG input file. This is enabled using a template of this file with a PLPROC function embedded in the template file.

File *.\param_files\milford1.wel.tpl* is a copy of the MFUSG input file *milford1.wel*. However an embedded PLPROC function is inserted into this file at its third and fourth lines. The first part of this file is reproduced below.

```
1145 50 AUTOFLOWREDUCE  IUNITAFR 166
 1139  0  6    Stress Period 1
$#p  replace_column(plist=well_rate;startpos=15;endpos=32,          &
$#p                 intidcolumn=1,tablelength=1139)
 126809        5.588000e+01
 126810        5.599000e+01
 126811        5.599000e+01
 126812        5.588000e+01
 126813        5.566000e+01
etc
```

The embedded function instructs PLPROC to record the values of certain elements of the *well_rate* PLIST between character positions 15 and 32 on each of the 1139 lines that follow. In doing so, it must overwrite text which already occupies these positions. So which elements of this PLIST are to be transferred? These are identified by the model cell indices that PLPROC reads from column 1 of this file. (The correspondence between *cl_mod* CLIST elements and MFUSG grid nodes was made when PLPROC read the MFUSG grid specification file at the start of the script.)

The *tablelength* argument in the above function is important. After writing 1139 lines in the manner directed by the function (thereby replacing existing lines), PLPROC transfers the remaining lines of the template file to the model input file that it writes. These lines contain extraction rates from Milford production wells.

3.3.10 River Properties

PLPROC's handling of RIV properties is similar to the above. However it is slightly more complicated because of the number of separate stream segments to which interpolation must take place, and because a MFUSG RIV input file is slightly more complex than a MFUSG WEL input file.

First the *work3d* PLIST is re-initialized.

```
work3d=1.0
```

Then, through a series of function calls like the following, river conductance multipliers are interpolated from a subset of *pp_riv* pilot points which belongs to a certain stream segment, to those model cells which belong to the same segment. Interpolation is through inverse squared distance. The first of 14 function calls recorded in file *plproc.dat* follows.

```
work3d(select=(riv_reach.eq.1))=ppr_mul.ivd_interpolate_2d(       &
                                 select=(ppr_reach==1),           &
                                 transform='log',                 &
                                 inv_power=2.0,                   &
                                 min_points=2,                    &
                                 max_points=100,                  &
                                 search_radius=2000.0)
```

Once these 14 function calls have been made, the MFUSG-based *riv_cond* PLIST (read from file *model_info.ndt* at the beginning of the script) is updated through application of these interpolated multipliers.

```
riv_cond=riv_cond*work3d
```

The process is then repeated for numbers that must be added to river elevations. First *work3d* is re-initialized.

```
work3d=0.0
```

Then, for each reach, numbers that must be added to river elevations are interpolated from *pp_riv* pilot points to reach-specific cells of the model grid. The first of 14 function calls follows.

```
work3d(select=(riv_reach.eq.1))=ppr_add.ivd_interpolate_2d(       &
                                 select=(ppr_reach==1),           &
                                 transform='none',                &
                                 inv_power=2.0,                   &
                                 min_points=2,                    &
                                 max_points=100,                  &
                                 search_radius=2000.0)
```

The riverbed PLIST is modified through addition. Then a new PLIST which holds riverbed bottom elevations is defined by subtraction of 1 ft from river elevations. (In retrospect, 1 ft is too small. I had forgotten that I was not working in metres.)

```
riv_head=riv_head+work3d
riv_bot =riv_head-1.0
```

Riverbed conductances, as well as river levels and bottom elevations are now recorded in the pertinent model input file. This file is named *milford1.riv*. PLPROC uses a template of this file named *milford1.riv.tpl* to write this model input file. This template file contains an embedded PLPROC function. The top part of *milford1.riv.tpl* is reproduced below.

```
# MODFLOW-USGs River Package
 1783 50
 1783 0    Stress Period 1
$#p  replace_column(plist=riv_head;startpos=10;endpos=27,           &
$#p                plist=riv_cond;startpos=30;endpos=47,           &
$#p                plist=riv_bot; startpos=50;endpos=67,           &
$#p                intidcolumn=1,tablelength=1783)
 19684     252.728308         1.080000e+01         251.728308 0
 20004     250.735800         7.350000e+00         249.735800 0
 33796     235.829573         9.075000e+03         234.829573 0
 33797     235.768248         3.757500e+04         234.768248 0
 33798     235.725263         3.750000e+04         234.725263 0
 33799     235.687421         3.885000e+04         234.687421 0
 33800     235.649552         3.960000e+04         234.649552 0
etc
```

The template-file-embedded *replace_column()* function informs PLPROC where it must write values of elements of the *riv_head*, *riv_cond* and *riv_bot* PLISTS on each ensuing line. Element numbers are read from the first column.

Back in *plproc.dat*, the function that instructs PLPROC to write *milford1.riv* using the *milford1.riv.tpl* template file concludes the PLPROC script.

```
write_model_input_file(template_file='.\param_files\milford1.riv.tpl',   &
                       model_input_file= 'milford1.riv')
```

4. Calibration Dataset

4.1 Overview

As is explained in the worked example report, the Milford model is history-matched against four different data types. These are:

- heads in observation wells;
- inflows/outflows to RIV reaches;
- pumping rates from extraction wells; and
- outflow through DRN boundary conditions.

Each of these is now described.

Files that hold observation-related data are stored in the *.\observ_files* subfolder of the *calibration* folder (and of other working folders that are supplied with this document).

4.2 Heads

When PEST runs the Milford model, it informs the operating system that it must run a batch file named *model.bat*. After running MFUSG, *model.bat* runs the USGMOD2OBS utility. USGMOD2OBS is a member of the PEST Groundwater Utility suite. It reads a binary "system state file" recorded by MFUSG. In the present case it reads file *milford1.hds*. It undertakes spatial and temporal interpolation of heads that it finds in this file to the sites and times of head measurements that are recorded in a so-called "site sample file" – in this case *.\observ_files\obshead.ssf*. (This file must be prepared by the modeller.) It then writes its own site sample file (named *modhead.ssf* in the present instance) which contain model-generated counterparts to observed system states. Responses to USGMOD2OBS's prompts are contained in file *usgmod2obs.in*. When USGMOD2OBS is run from *model.bat* it is directed to these responses through its command line:

```
.\exec\usgmod2obs < usgmod2obs.in
```

Time-interpolation from model output times to measurement times is linear. (This is hardly a difficult procedure in the present case as there is only one model output time and only one measurement time.) Spatial interpolation from the model grid to the sites of observation wells relies on interpolation factors supplied in an interpolation factor file. This file is named *.\observ_files\wells_interp.dat*. Where a MFUSG grid is regular, or fashioned from a regular grid through quadtree refinement of that grid, these interpolation factors can be calculated using the USGQUADFAC utility. If run from the *.\observ_files* subfolder, USGQUADFAC prompts and responses are as follows.

```
Enter name of unstructured grid specification file: ..\param_files\milford1.gsf
 - file ..\param_files\milford1.gsf read ok.
Are grid cell boundaries N-S and E-W? [y/n]: y

Enter name of unstructured MODFLOW-USG discretization file: ..\milford1.dis
 - file ..\milford1.dis read ok.

Enter name of bore coordinates file: bores.crd
 - 114 bores and coordinates read from bore coordinates file bores.crd

Enter name of bore listing file: bores.crd
 - 114 bores read from bore listing file bores.crd

Enter name for node-to-bore interpolation file: wells_interp.dat

Enter a suitable value for epsilon: 1e-3
```

File *bores.crd* mentioned in the above script is another user-prepared file. It records coordinates of observation wells, as well as model layers to which they belong.

4.3 Water Exchange with Rivers

model.bat deploys the USGBUD2SMP program supplied with the PEST Groundwater Utilities to read cell-by-cell water exchange between the flow domain and RIV boundaries from the MFUSG-generated cell-by-cell flow term file *milford1.cbb*. It amalgamates cell-by-cell flows into reach inflows/outflows, and then records these summed flows in a site sample file named *modriv.ssf*. These flows can be compared with observed flows recorded in *.\observ_files\obsriv.ssf*. Normally, the SMP2SMP utility would be run after USGBUD2SMP in order to undertake time-interpolation from model output times to observed times before this comparison is made. However this is not required when there is only one output time. If necessary, SMP2SMP also re-orders reaches in the model-generated site sample file so that they are in accordance with ordering provided in the observed site sample file. This is not required either; *obsriv.ssf* was constructed to respect ordering of reaches in the USGBUD2SMP output file.

Keyboard responses to USGBUD2SMP's prompts are contained in file *usgbud2smp_riv.in*. One of these responses refers to a file named *.\observ_files\river_obs_reaches.dat*. This file contains a single column of data which associates an integer with every node of the Milford model grid. These integers are mostly zero. Non-zero values identify cells to which a RIV boundary condition is assigned; they also denote the observation reach number of each RIV-affected cell. ("Observation reaches" should be distinguished from "parameter reaches"; see above.) USGBUD2SMP keyboard input ascribes a name to each of these reaches. These names are "reach*N*" where *N* is the reach number.

4.4 Pumping Rates

As is described above, MFUSG automatically reduces extraction rates from pumping wells in order to prevent them from going dry. Actual extraction rates are read from *milford1.cbcln* using another instance of USGBUD2SMP; keyboard input is provided by *usgbud2smp_pump.in*. CLN "zones" are defined in *.\observ_files\clns.dat*. The name of an extraction well is associated with each of these zones through the pre-recorded keyboard input file *usgbud2smp_pump.in*. USGBUD2SMP records well extraction rates in site sample format in file *modpump.ssf*. Meanwhile "observed" pumping rates are supplied by the user in file *.\observ_files\obspump.ssf*. These are, of course, the pumping rates supplied to these same wells through the MFUSG WEL package input file *milford1.wel*.

As for river water exchanges, running of the SMP2SMP utility after USGBUD2SMP in order to implement time interpolation from model output times to measured times is foregone. This simplifies *model.bat*. Its omission is justified by the steady state nature of the model, and because the ordering of wells in *.\observ_files\obspump.ssf* coincides with that in *modpump.ssf*.

4.5 DRN Outflow

The USGBUD2SMP utility is run yet again in *model.bat*, this time to accumulate DRN outflow through each of the "drain reaches" depicted in the following map.

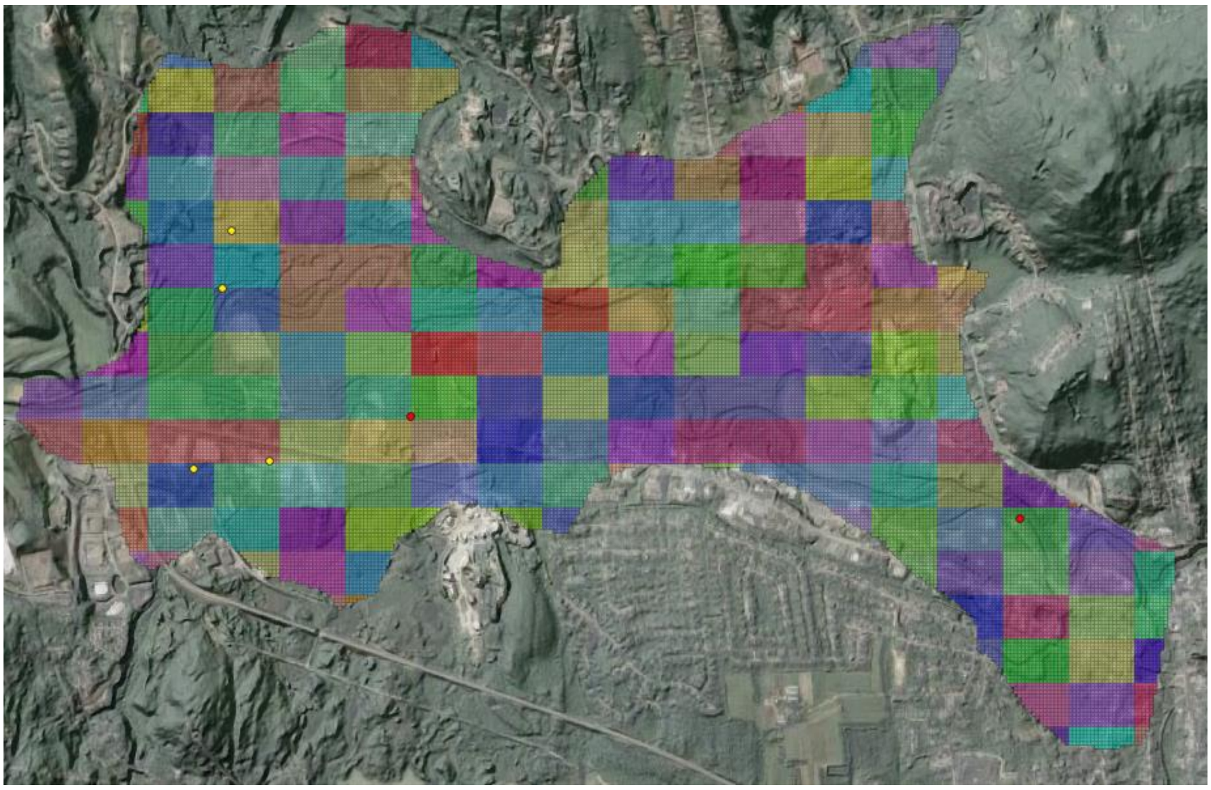

Figure 4.1. "Drain reaches" used by the Milford model.

Reaches are defined in *.\observ_files\drn_obs_reaches.dat*. This file contains a single column of data in which an integer is assigned to every model cell. Non-zero integers define drain reach numbers. The keyboard response file *usgbud2smp_drn.in* associates a name with each reach; this "name" is the same as its reach number. USGMOD2SMP records flows from each reach in the site sample file *moddrn.ssf*. These must be compared with "observed" DRN flows that are recorded in *.\observ_files\obsdrn.ssf*. Observed outflows from all DRN reaches are zero; hence file *.\observ_files\obsdrn.ssf* is easy to prepare.

File *.\observ_files\drn_obs_reaches.dat* which assigns cells in model layer 1 to the patchwork quilt of drain reaches depicted in Figure 4.1 was produced by a self-written program. Preparation of the keyboard input file *usgbud2smp_drn.in* was manual – and a little painful.

5. Preparing PEST_HP Input Files

5.1 General

This document does not provide detailed instructions on how to build the PEST control file *milford1.pst* and template/instruction files cited therein. *milford1.pst* is a text file; it can be assembled using a text editor. However this is a somewhat tedious and error-prone way to build it. Fortunately, assistance in building PEST input datasets is available through utility programs supplied with PEST, and with the PEST Groundwater Utility suite. Pertinent utilities are mentioned in this chapter. Refer to their documentation for further details.

5.2 Parameters

5.2.1 Template Files

Template files are usually prepared with a text editor. They are modified from model input files; their primary design specification is that when parameter spaces are replaced by parameter values, the resulting model input file has integrity as far as the model is concerned.

For the Milford model, all parameters are associated with pilot points. All template files are therefore modifications of PLPROC input files. These template files are named after the PLPROC input files that they are used to write. They are:

- *ppset1.dat.tpl*
- *ppset2.dat.tpl*
- *ppset3.dat.tpl*
- *pp_rivers.dat.tpl*
- *pp_wells.dat.tpl*

Template files cite the names of parameters. A parameter name, together with two "parameter delimiters", identifies the space to which that parameter's value must be written on a model input file. As is explained in PEST documentation, PEST requires that parameter names be 12 characters or less in length. This limit does not apply to programs of the PEST++ suite; they can accommodate parameter names of any length.

Ideally, the spaces in which parameter values are written in model input files should be 14 characters or more in length. This allows their representation with a high level of numerical precision; this helps to preserve the integrity of finite-difference derivatives calculation.

5.2.2 Commencing Construction of PEST Control File

The TPL2PST utility that is supplied with PEST reads one or a number of template files. It creates an embryonic PEST control file in which all parameters that are cited in all of these template files are featured. Group names, initial parameter values and parameter upper and lower bounds are also recorded in this file. (These are provided to TPL2PST through its user-prepared control file.) The PEST control file which TPL2PST writes will require user-editing, but it is a good start.

Initial parameter values that are provided in a PEST control file should also be preferred parameter values. Meanwhile, parameters of different types, and parameters pertaining to different layers, should be assigned to different parameter groups. This facilitates the introduction of regularisation to a PEST control file.

As explained in PEST documentation, the primary role of parameter groups is to define the way in which finite-difference derivatives are calculated. However when the ADDREG1 or ADDREG2 utilities are used to add regularisation to a PEST control file, prior information equations that embody preferred parameter values are assigned to observation groups whose names are built from the names of pertinent parameter groups; the string "*regul_*" is prefixed to the parameter group name to form the observation group name. If the resulting observation group name is greater than 12 characters in length, its back end is truncated. This may result in nonunique observation group names. Hence it is wise to restrict parameter group names to six characters or less in length so that observation group names are still unique when prefixed with "*regul_*".

Also, by assigning different observation group names to prior information equations which describe parameters of different types and/or that belong to different model layers, PEST has the opportunity to apply differential regularisation weighting to these different parameter types. This may result in more pleasing calibrated parameter sets.

The control file that is written by TPL2PST includes a "control data" section. Default values of control variables that are recorded in this section are good for most occasions. The TPL2PST-produced PEST control file also includes part of a "model input/output" section in which the names of template files, and the model input files that they write, are featured. The names of the latter are provided by the user in the TPL2PST input control file.

5.2.3 Special Considerations for River Elevation Parameters

Parameters whose initial values are zero can sometimes cause problems during the inversion process, especially when parameter change limits are enforced. Parameters belonging to the *rhadd* parameter group are of this type. The values of these parameters are added to the elevations of RIV boundary conditions. A few measures were taken in file *milford1.pst* (possibly more than are actually needed) to forestall problems that parameters such as these may precipitate.

First, note that all of these parameters are tied to the first of them (i.e. to parameter *ppr_add1*) in *milford1.pst*. As is described in the Milford worked example report, this strategy was adopted during calibration of the Milford model using PEST_HP. However when uncertainty analysis was undertaken using PESTPP-IES, all of these parameters were fixed. This was done after an examination of PEST_HP-calculated sensitivities revealed that, even when varied as a block, these parameters have little effect on model outputs.

Steps taken to ensure good behaviour of parameter *ppr_add1* during the inversion process are as follows.

- It is given an OFFSET of -10.0. Its initial value is 10.0. Therefore, the value that the model sees is 0.0.
- Rather than being assigned a factor or relative change limit, it is assigned an absolute change limit. In the "control data" section of the PEST control file, this limit is declared to be 1.0 ft.
- In the "parameter groups" section of file *model1.pst*, parameters belonging to the *rhadd* parameter group are endowed with an absolute derivative increment (of 0.1 ft) rather than a relative increment.

5.3 Observations

5.3.1 General Observations

The PESTPREP2 utility supplied with the Groundwater Utility suite can be used to fill the "observation groups" and "observation data" sections of a PEST control file whose construction was initiated using TPL2PST. Use of PESTPREP2 is predicated on the assumption that observations that model outputs must match are recorded in site sample files. It is also assumed that, through use of model postprocessing programs such as USGMOD2OBS, USGBUD2SMP, SMP2SMP and other members of the Groundwater Utility suite, the model that is run by PEST (i.e. the composite model that is encompassed in a batch file such as *model.bat*) records model-generated counterparts to these observation site sample files.

If TPL2PST is not employed to initiate construction of a PEST control file, then PESTPREP1 (from the Groundwater Utility suite) can be used to commence PEST control file construction, and to provide the new control file with reasonable default values for control variables. It can also be used to initiate the process of adding observation data to a PEST control file. Whether PEST control file construction is initiated using TPL2PST or PESTPREP1, PESTPREP2 can be run as many times as is necessary to complete the process of adding observation data to a PEST control file. PESTPREP2 reads a single observation site sample file and its model-generated counterpart. It then performs the following tasks.

- It writes an instruction file that reads the model-generated site sample file; observations are named in the process.
- It adds observations to the "observation data" section of the PEST control file; observed values are read from the observation site sample file.
- It adds a new observation group to the "observation groups" section of the PEST control file; the name of this group is supplied by the user.
- It adds the name of the instruction file, and of the model output file that the instruction file must read, to the "model input/output" section of the PEST control file.

The PEST control file that the above programs collectively write may require some modifications. For example, a user may desire that subsets of observations that feature in a single site sample file be assigned to separate observation groups.

PESTPREP2 endows all observations with a weight of 1.0. The ADJOBS utility (from the Groundwater Utility suite) can reformulate weights as functions of observation values. ADJOBS reads an existing PEST control file and writes a new one containing reformulated weights. Alternatively, or as well, a user may wish to set some weights manually in order to reflect the credibilities of individual observations. We will describe shortly how weights can be adjusted on an observation group by observation group basis so that the contribution to the objective function made by each observation group is the same as that of any other observation group. Hence, whether weights are adjusted manually or by using ADJOBS, relatively of weights within each observation group (rather than their actual values) should be the most important weights-setting criterion at this stage of the overall weights assignment process.

Note that weights ascribed to observations *pump_keys_1* and *pump_savage_1* are set to zero in file *milford1.pst*. These are pumping rates for the Savage and Keyes wells. These wells do not pump during the time over which the model is calibrated.

5.3.2 A Special Observation

The Milford worked example report explains why groundwater inflow into observation reach 2 of the stream network requires special attention. The downstream gauging station for this reach is outside the model domain. "Measured" steady state inflows to this reach therefore exceed that which should be simulated by the model. Hence model-calculated flow into this reach should induce an objective function penalty only if it exceeds the "measured" inflow, but not if it is less than the measured reach inflow.

PEST and BEOPEST make no provision for one-sided observations. (However one-sided observations can be accommodated by using the OBS2OBS utility supplied with PEST as a model postprocessor.) PEST_HP does, however, support them. If an observation belongs to an observation group whose name begins with "<@", then PEST_HP does not penalize the model output value if it is less than the observed value; in fact it migrates the modelled value up to the observed value so that the residual is zero. In contrast, if the model-generated value is greater than the observed value, a residual is calculated in the normal way; the weighted residual then contributes to the objective function. The opposite behaviour is enforced for observations that belong to groups that begin with ">@".

An inspection of file *milford1.pst* reveals that inflow into reach 2 of the stream network belongs to an observation group named *<@river*.

Members of the PEST++ suite also support one-way observations. However the naming protocol for observations of this type is different. This is discussed later in this document.

5.4 Model Batch File

The "model" which is run by PEST is usually a series of commands encapsulated in a batch file. For the Milford model, the batch file is named *model.bat*. Inspection of this file reveals that it undertakes the following sequence of tasks.

- All files that are written by one program and read by another program are deleted at the start of the processing sequence. Hence if any link in the processing chain is broken, the ensuing program does not read an old output file of the preceding program, mistaking it for a new one. It crashes instead.
- PLPROC is run in order to write MFUSG input files whose contents are dependent on values assigned to parameters.
- MFUSG is run.
- Observation postprocessing is undertaken. USGMOD2OBS and a sequence of instances of USGBUD2SMP are run. All of these programs read binary files written by MFUSG. They extract information from which model-generated counterparts to field measurements can be calculated. They record the latter in site sample files (see above).

5.5 Balancing Weights

At this stage of assembling a PEST input dataset, PESTCHEK should be run to check that all aspects of it are correct and consistent. Then the NOPTMAX variable in the "control data" section of the PEST control file should be set to zero. PEST should then be run. With NOPTMAX set to zero, PEST runs the model once. It then records on its run record file the objective function, as well as contributions made to the total objective function by all observation groups. It then ceases execution.

Normally, after completion of this initial model run, it is found that the contribution made to the overall objective function by some observation groups is much greater than that made by other observation groups. One reason for this is that some observations (and their model-calculated counterparts) are of very different numerical magnitude than other observations. This effect can be partly alleviated by ensuring that weights reflect measurement noise. Larger-valued observations are normally accompanied by larger-valued measurement errors; their weights should therefore be smaller.

However, measurement noise is not the only consideration when assigning weights. A case can also be mounted (based on the nature of so-called "structural noise" that is often responsible for the bulk of model-to-measurement misfit) that each component of a multi-component objective function should have roughly equal visibility in the total objective function at the commencement of an inversion process. This argument is especially strong if observation groups that define these components host data of different types. A weights assignment strategy that promotes rough equality of objective function component visibility ensures that information that is resident in each observation group is able to influence the values of estimated parameters.

Primitive weights balancing can be readily achieved using the PWTADJ1 utility supplied with PEST. This utility should be run after PEST has been run with NOPTMAX set to zero in the manner described above. PWTADJ1 reads objective function components from the run record file that is written by PEST.

It then revises weights in the PEST control file, writing a new PEST control file in the process. Objective function components that arise when PEST is run using this new PEST control file are roughly equal.

PWTADJ1 is run using a command such as:

```
pwtadj1 milford1 milford2 1000
```

This creates a new PEST control file named *milford2.pst* in which the objective function component contributed by each observation group is 1000.

Note that, if desired, inter-group weights balancing can be postponed until after regularisation is added to the PEST control file; PWTADJ1 ignores regularisation.

5.6 Adding Regularisation

Before adding regularisation to a PEST control file, parameters which need to be fixed and tied should be designated as such. When the Milford model was calibrated, recharge parameters were fixed; they are later released when undertaking uncertainty analysis. At the same time, pilot point parameters which add/subtract elevation to the river level were tied together, thereby constraining them to behave as a single parameter.

The ADDREG1 and ADDREG2 utilities add "preferred value" Tikhonov regularisation to a PEST control file. The preferred value ascribed to each parameter is its initial value as recorded in the "parameter data" section of the PEST control file. Hence care should always be exercised when building a PEST control file to ensure that initial parameter values are "expected" parameter values (i.e. mean parameter values from the point of view of the prior parameter probability distribution).

Differences between ADDREG1 and ADDREG2 include the fact that ADDREG2 allows a user to nominate the target measurement objective function on its command line. When calibrating the Milford model, the target measurement objective function was set to a very low value, namely the ADDREG1 default value of 1E-10. However, as is explained in the worked example report, parameters emerging from only the third iteration of the inversion process were deemed to "calibrate" the model.

ADDREG1 undertakes the following tasks.

- It creates a prior information equation for every adjustable parameter. Each equation sets the preferred value of a parameter to its initial value. A weight of 1.0 is assigned to each such prior information equation. Each equation is assigned to an observation group that is named after the parameter group to which the cited parameter belongs; a prefix of "*regul_*" is appended to this observation group name.
- It adds the names of these new observation groups to the "observation groups" section of the PEST control file.
- It alters the value of the PESTMODE control variable to "regularisation".
- It adds a "regularisation" section to the end of the PEST control file.

5.7 Covariance Matrices

5.7.1 Overview

As stated above, ADDREG1 assigns a weight of 1.0 to all prior information equations that it introduces to a PEST control file. However a better strategy for a group of prior information equations that cite spatial parameters of the same type is that they be assigned a covariance matrix instead of weights. A covariance matrix "suggests" to the inversion process that, should heterogeneity need to arise, it does so in a way that involves neighbouring groups of pilot points rather than individual pilot points. This avoids "bullseyes" and other unsightly features in the estimated parameter field.

To assign a covariance matrix instead of weights to a particular regularisation group (i.e. an observation group that is associated with a set of prior information equations that implement Tikhonov regularisation), the name of the file which holds the covariance matrix should be recorded after the name of the respective group in the "observation groups" section of the PEST control file.

The PEST Groundwater Utility suite provides a number of programs that automate construction of covariance matrices for pilot point parameters (or any parameters, for that matter, for which spatial coordinates are supplied). These are PPCOV and PPCOV_SVA for two-dimensional spatial parameters, and PPCOV3D and PPCOV3D_SVA for three-dimensional spatial parameters. (Note that the Milford model employs only two-dimensional parameters.) All of these programs base their construction of a covariance matrix on a variogram. Programs with an "SVA" suffix allow the properties of that variogram to vary in space. These latter programs were used to construct covariance matrices for the Milford model.

We now describe construction of file *ppset1_p0625.cov*. This file contains the covariance matrix that is assigned to the *regul_kx2bas* observation group ("bas" stands for "basic"). This is used for regularisation of layer 2 hydraulic conductivity parameters that are based on the *ppset1* family of pilot points. Construction of other covariance matrices follows a similar trajectory.

Note that, as described in the Milford worked example report, covariance matrices that are used for regularisation are also used for uncertainty analysis.

5.7.2 Statistical Specification File

Construction of *ppset1_p0625.cov* requires two steps. First the MKPPSTAT utility (supplied with the PEST Groundwater Utility suite) is run in order to build an input file for PPCOV_SVA. This input file is known as a "statistical specification file"; more on this below. MKPPSTAT prompts and responses are as follows.

```
Enter name of pilot points file: ppset1.pts
 - data for 32 pilot points read from pilot points file ppset1.pts

Enter no. of pilot points to compute local ave. pp. sepn.: 10
Enter factor of ave. separation for local variogram "a" value: 1.5

Enter name for pilot point statistical spec. file: ppset1.sta
- file ppset1.sta written ok
```

The "pilot points file" that MKPPSTAT requests follows legacy protocols that have been used by members of the PEST Groundwater Utility suite for years. The figure below shows the first part of *ppset1.pts*.

```
1      973542.472930681    124898.592052393      1    150.0
2      975379.100994654    124531.266439599      1    150.0
3      975033.382770847    122932.319654493      1    150.0
4      975271.064049714    126346.287114584      1    150.0
5      976113.752220243    127232.190063088      1    150.0
6      974946.953214896    128312.559512484      1    150.0
7      975551.960106557    129522.573295807      1    150.0
etc
```

Its most important features are as follows.

- The file must have five columns. Even though the contents of the last column are ignored, it must nevertheless be present.
- The first four columns must provide (in order):
  - pilot point identifiers (integers or character strings);
  - pilot point *x* and *y* coordinates;
  - zones (identified by an integer) to which each pilot points belongs.

- There must be no column headers.

(Although they are not needed for running the Milford model, a set of **.pts* files corresponding to the five pilot point families used by that model are included in the *calibration\param_files* subfolder that is supplied with this document.)

Pilot point identifiers in **.pts* files need not be the same as those which identify pilot points in files that are read by PLPROC. However the order in which pilot points are listed in these files is important. This order must be the same as that in which parameters that are associated with each group of pilot points is recorded in the PEST control file. This will occur naturally if pilot points are listed in the same order in PLPROC input files as that in which they are listed in MKPPSTAT input files, and if TPL2PST or PESTPREP1 assists in construction of the PEST control file.

The first part of the statistical specification file that is written by MKPPSTAT is shown below.

```
point   x              y         zone  nugget  sill      a         hanis  bearing
 1     973542.473     124898.592  1     0.0     1.0      5420.724   1.0   0.0
 2     975379.101     124531.266  1     0.0     1.0      3988.680   1.0   0.0
 3     975033.383     122932.320  1     0.0     1.0      5301.863   1.0   0.0
 4     975271.064     126346.287  1     0.0     1.0      3789.500   1.0   0.0
 5     976113.752     127232.190  1     0.0     1.0      3530.614   1.0   0.0
 6     974946.953     128312.560  1     0.0     1.0      4331.875   1.0   0.0
 7     975551.960     129522.573  1     0.0     1.0      4806.344   1.0   0.0
 etc
```

As is apparent from the above figure, variogram specifications are attributed to every pilot point. This file can be user-edited if desired. This can be useful if, for example, it is desired that anisotropy be location-dependent (as may be required for alluvial deposits which follow a meandering stream). Meanwhile, the variogram range that MKPPSTAT attributes to any pilot point is dependent on local pilot point spatial density.

The sill of a variogram is equal to the variance of the parameter which it describes. Variance is the square of standard deviation. MKPPSTAT sets the variogram sill to 1.0 at the locations of all pilot points. These should be altered to reflect "true" parameter variances. PPCOV_SVA (which is run next) allows the variogram sill (and all other variogram properties) to vary with location. In the present case, however, the sill is assumed to be uniformly 0.0625 for all hydraulic conductivity parameters which are associated with the *ppset1* pilot point family. (See the worked example report for more details.) So entries in the "sill" column of the MKPPSTAT-generated statistical specification file should all be altered to this uniform value before running PPCOV_SVA.

5.7.3 Building the Covariance Matrix

Once the statistical specification file is ready, construction of the covariance matrix is a simple matter. It can be accomplished by running PPCOV_SVA, and responding to its prompts as follows:

```
Enter name of pilot points statistical specs file: ppset1_p0625.sta
Skip a line at the top of this file?  [y/n]: y
 - data for 32 pilot points read from pilot points file ppset1_p0625.sta

Enter minimum allowable separation for points in same zone: 0

Is overall variogram spherical, exponential or Gaussian? [s/x/g]: x

Enter name for output matrix file: ppset1_p0625.cov
Enter pilot point prefix for parameter name (<Enter> if none):<Enter>

Filling covariance matrix....
Using SVD to assure positive definiteness of matrix....
- file ppset1_p0625.cov written ok.
```

The "prefix for parameter name" prompt is worth a few words.

When covariance matrices are used for uncertainty analysis, elements of this matrix must be linked to parameters in a PEST control file. There are a number of ways to do this. One option is to endow rows and columns in a covariance matrix file with names that identify the parameters to which they pertain. For older members of the Groundwater Utilities, parameter names were derived from pilot point identifiers through addition of a short prefix. Members of the PPCOV suite can names rows and columns of the covariance matrix which they build accordingly.

We do not adopt this protocol in Milford modelling. Instead, we ensure that ordering of rows and columns in the covariance matrix is the same as ordering of parameters in the PEST control file. The first and last parameters (in the PEST control file) to which the matrix pertains are then used to link this matrix to a list of parameters when the covariance matrix is used in uncertainty analysis; see later.

Where a covariance matrix replaces weights assigned to prior information equations during regularised inversion, ordering is again important. ADDREG1 writes prior information equations in the same order as that in which it finds parameters in the “parameter data” section of the PEST control file. The ordering of parameters in these “preferred value” prior information equations is thus the same as that of parameters in the “parameter data” section of the PEST control file. By maintaining consistency in parameter ordering in all files that pertain to pilot points, linkages between parameters and covariance matrices that are used for regularisation purposes is automatic when the name of a covariance matrix file is recorded behind the name of a regularisation group in the “observation groups” section of a PEST control file.

5.8 Finishing Touches

5.8.1 General

The sixth line of file *milford1.pst* is reproduced below. Variables on this line govern parameter update testing using different values of the Marquardt lambda.

```
 10.0  -3.0  0.3  0.03  10 999  lamforgive  uptestmin=30
```

In contrast to PEST and BEOPEST, PEST_HP does not read the first five entries on this line. (Actually, it reads them but ignores them.) Its calculation of Marquardt lambda values is automatic. It is partly based on the number of parallelisation agents that it has at its disposal.

Nor does PEST_HP read “lamforgive”. This is automatically turned on.

This leaves only two variables on the above line that are of interest to PEST_HP. These are now discussed.

5.8.2 Broyden Updating

The sixth entry on line six of *milford1.pst* is “999”. This turns Broyden Jacobian updating on. With Broyden updating turned on, PEST_HP undertakes two rounds of “lambda testing” during each iteration of the inversion process. Just after it completes the first round of lambda testing, it upgrades the current Jacobian matrix using model outputs that were calculated as lambda-based parameter upgrades were tested. Lambda testing is then repeated; however parameter upgrades are now calculated using the updated Jacobian matrix. In difficult calibration contexts this can improve the efficiency of the inversion process. In the present case, however, it does not help much, as calibration of the Milford model is not a numerically difficult undertaking. (Nevertheless, it is good practice to activate Broyden updating, as one never knows ahead of time whether it will be useful or not. There is one thing for sure; it will do no harm.)

5.8.3 UPTESTMIN

PEST_HP’s implementation of Marquardt-lambda-based parameter upgrade testing (i.e. “lambda-testing”) is more complex than that of PEST. As well as testing a suite of different Marquardt lambda values, PEST_HP also undertakes a basic line search along each lambda-defined direction in parameter space. The number of Marquardt lambdas that it employs, and the number of points along each lambda-defined parameter upgrade direction that it tests, are both dependent on the number of parallelisation agents to which it has access. At the bare minimum, PEST_HP employs three lambda values, and selects one point along each lambda-defined parameter upgrade direction. It does this even if it has less than 3 parallelisation agents at its disposal.

The “uptestmin=30” string on the sixth line of *milford1.pst* instructs PEST_HP that it must undertake its lambda selection and line search strategy as if it had 30 parallelisation agents at its disposal. Experience shows that this makes maximum use of the Jacobian matrix that it has gone to so much trouble to fill.

Note that PEST/BEOPEST will object with an error message if you try to run these programs using a PEST control file in which UPTESTMIN (and other PEST_HP-specific control variables) are activated. However PEST will not object if NOPTMAX is set to zero, and if it is run using the “/hpstart” switch. See the PEST_HP manual for further details.

5.8.4 The PARSAVEITN Variable

The PARSAVEITN variable is activated on the last line of the “control data” section of *milford1.pst*. It informs PEST_HP that it must save a parameter value file at the end of every iteration of the inversion process. This is in addition to the parameter value file that it saves (and continually updates) in which best parameters achieved up to any point in the inversion process are recorded. The name of the latter parameter value file is *case.par*, where *case* is the filename base of the PEST control file. The names of the former set of parameter value files are *case.par.N*, where *N* is the iteration number.

After setting NOPTMAX to 50, the *milford1.pst* PEST control file is ready for use by PEST_HP.

5.9 Running PEST_HP

PEST_HP cannot undertake model runs in serial. It can only run the model with the help of at least one parallelisation agent. To run PEST_HP with a single agent, open two command line windows. Note that you can open a second command line window in a folder to which a command line window is already open by typing the command:

```
start cmd
```

In one of these windows start the PEST_HP manager. You can use the normal version of PEST_HP:

```
pest_hp milford1 /h :4004
```

Or you can use the version of PEST_HP that links to the Internal Maths Kernel Library. This performs matrix operations such as SVD very quickly indeed.

```
pest_hp_mkl milford1 /h :4004
```

In the other window start the PEST_HP parallelisation agent:

```
agent_hp milford1 /h %computername%:4004
```

You can substitute the computer’s hostname or IPv4 address for `%computername%` if you wish.

To use multiple agents, multiple copies of the working folder and all its subfolders must be made. The command to start AGENT_HP must be issued from each command line window that is open to an agent’s working folder. Agent folders can reside on any computer that has network contact with the computer on which the manager is running. If an agent is initiated on another computer, then you must use the manager computer’s IP address or hostname instead of `%computername%` when starting it.

5.10 When PEST_HP has Finished

As is discussed in the Milford worked example report, the parameter field calculated at the end of the third iteration of the inversion process is deemed to "calibrate" the model. Parameter values attained in this way can be inserted into a new PEST control file named *milford1_soln.pst*. This is done using the PARREP utility supplied with PEST. The command is:

```
parrep milford1.par.3 milford1.pst milford1_soln.pst
```

If NOPTMAX is set to 0 in *milford1_soln.pst*, then PEST_HP runs the model once if invoked using the following command.

```
pest_hp milford1_soln /h :4004
```

The command to run the agent is:

```
agent_hp milford1_soln /h %computername%:4004
```

Once PEST_HP has completed this model run, it shuts down after recording a run record file, as well as a few other files. Meanwhile, model input files will contain calibrated hydraulic property values while model output files will contain numbers that are calculated using these values. Objective functions recorded in the run record file (*milford1_soln.rec*), and residuals recorded in the residuals file (*milford1_soln.res*) will also reflect the model's use of calibration-estimated parameter values.

The following should be noted.

- The regularisation objective function recorded in *milford1_soln.rec* will not be the same as in iteration 3 of the previous PEST_HP run, as PEST_HP does not calculate regularisation weight factors when it is asked to commission just a single model run.
- If you use PEST rather than PEST_HP to perform this single model run, then the residual associated with the *riv_reach2_1* observation will be different from that calculated by PEST_HP. This is because PEST does not support one-sided residuals. Observation groups whose names begin with "<@" or ">@" are treated just like any other observation group. The total objective function calculated by PEST will therefore be different from that calculated by PEST_HP.

6. Preparing for PESTPP-IES

6.1 Overview

We now turn to the use of PESTPP-IES in obtaining a suite of parameter fields, all of which fit the calibration dataset. There is a new working folder; it is named *ies*. As is discussed in the Milford worked example report, our starting point in using PESTPP-IES is the “calibrated” parameter set. Calibrated parameter values are listed as initial parameter values in file *milford1_soln.pst* situated in the *calibration* folder, which has been our working folder up until now.

We begin this section by describing how to construct a new PEST control file for the use of PESTPP-IES. This file is modified from *milford1_soln.pst*. *milford1_soln.pst* has been copied from the *calibration* folder to the *ies* folder as *work1.pst*.

6.2 Preparing the PEST Control File

6.2.1 Removing Regularisation

Uncertainty analysis is the opposite of regularised inversion. The latter seeks the heterogeneity that *must* exist to explain an observation dataset. The former seeks the heterogeneity that *may* exist that is compatible with an observation dataset. Numerical regularisation must therefore be removed from a PEST control file that forms the basis for posterior uncertainty analysis. This can be done using the SUBREG1 utility that is supplied with PEST. The command is:

```
subreg1 work1 work2
```

When run using this command, SUBREG1 creates a new PEST control file named *work2.pst*.

6.2.2 Unfixing and Untying Parameters

Recharge parameters were fixed during calibration of the Milford model. However, they are adjustable during uncertainty analysis. Hence the transformation status of all of them must be altered from “fixed” to “log”.

In contrast, parameters that modify the elevations of RIV boundaries are fixed for uncertainty analysis. Hence the transformation status of all of them must be altered from “none” or “tied” to “fixed”. At the same time, their “absolute(1)” change limit types should be altered to “relative”. (Naturally, the type of change limit does not matter if a parameter does not change. However PESTPP-IES does not recognize “absolute(1)”). The list of tied/parent parameter pairs must also be removed from the second half of the “parameter data” section of the PEST control file.

The “uptestmin=30” string should be removed from the sixth line of *work2.pst*. PESTPP-IES does not need this.

work3.pst is a PEST control file in which all of these changes have been made.

6.2.3 Removing Some Observations

Weights for observations *head_204_1* and *head_78_1* should be set to 0.0. The reason for removal of these observations from the history-matching dataset is discussed in the Milford worked example report. See *work4.pst*.

6.3 Alteration to PLPROC

As was described in the worked example report, an upper limit of 450 ft/day is imposed on all hydraulic conductivities written to Milford model input files. This limit is imposed in PLPROC. The following lines have been added to file *plproc.dat* just above those which instruct PLPROC to record hydraulic conductivities on model input files (around line 106).

```
kx1=min(450,kx1)
kx2=min(450,kx2)
kx3=min(450,kx3)
```

6.4 Generating Realisations of Observation Noise

6.4.1 Background

Two features of a history-matching dataset contribute to the posterior uncertainties of model parameters. These are:

- noise associated with measurements comprising the dataset; and
- absence of information in the dataset.

Combinations of parameters that are not informed by a history-matching dataset retain their prior probability distributions. On the other hand, estimates of parameter combinations that are informed by a history-matching dataset are contaminated by the fact that field measurements are a little bit "wrong" thanks to accompanying measurement noise.

To mimic the action of measurement noise, PESTPP-IES estimates posterior parameter realisations using real-world measurements that are augmented with realisations of measurement noise. Hence for each parameter realisation that comprises a parameter ensemble, PESTPP-IES (optionally) adds a realisation of measurement noise to observations featured in the "observation data" section of the PEST control file in order to generate a complementary observation ensemble. Alternatively a modeller can provide PESTPP-IES with realisations of noise-enhanced observations him/herself. If PESTPP-IES generates its own realisations of measurement noise, it assumes that the observation weight associated with each observation in the "observation data" section of a PEST control file is equal to the standard deviation of noise associated with that observation. Random number generation is then quite straightforward.

In the present case, we generate our own ensemble of noise-enhanced observations rather than ask PESTPP-IES to do this. The reasons for this will become clear in a moment. In order to do this, we need a PEST control file in which observation weights are the inverse of observation noise standard deviations. These "correct" weights will be back-calculated from model-to-measurement misfit attained through the previous calibration process. Of course, measurement error is not the only reason for misfit. Conceptual error in the steady state assumption, and model imperfections, also contribute to misfit. For convenience, we treat the totality of all of these contributions as "measurement error", or "measurement noise".

When we run PESTPP-IES, however, we will retain the same weighting scheme as that which we used when calibrating the Milford model using PEST_HP. Recall that these weights were calculated with balancing of information from different sources in mind. Because we retain these weights, we must tell PESTPP-IES not to calculate realisations of noise-enhanced observations (which, as explained above, requires use of these weights). We will do this shortly.

We now outline the steps that are required to obtain an ensemble of noise-enhanced observations.

6.4.2 Generating "Correct" Weights

The Milford model can be run by running PEST_HP using the *work4.pst* PEST control file. NOPTMAX is set to zero in this file, so PEST_HP runs the model once, records objective function components in its run record file (*work4.rec*), and then ceases execution. Type:

```
.\exec\pest_hp work4 /h :4004
```

in one command-line window open to the *ies* working folder. Type:

```
.\exec\agent_hp work4 /h %computername%:4004
```

in another window which is open to this same folder.

Now use the PWTADJ2 utility to generate a new PEST control file in which weights are "correct". The command is:

```
 pwtadj2 work4 work5 g
```

work5.pst is the new PEST control file.

6.4.3 Generating Noise-Enhanced Observations

We can now generate randomised observations based on "correct" weights. PESTPP-IES will be informed that it must read these noise-enhanced observations from either a JCB file or a CSV file. A JCB file is a binary file which stores a matrix. The rows of the "matrix" in this case would simply be realisations of noise-enhanced observations. We use the CSV alternative in the present case. The utility which does the work is RANDOBS; this is supplied with the PEST suite. It is run as follows.

```
 Enter name of observation weights file: work5.pst
 Enter factor to apply to all weights: 1

 How many realizations to generate? 500
 Include initial observations as base realization? [y/n]: n

 Enter name for output file: randobs.csv
 Realizations are rows or columns in csv file? [r/c]: r

 Enter integer random number seed (<Enter> if default): <Enter>
 - reading file milford2.pst...
 - 307 observations read from file milford2.pst.
 - generating realizations and writing file weights.csv...
 - file weights.csv written ok.
```

Note that if you run RANDOBS yourself, the contents of file *randobs.csv* will be slightly different from those that appear in the version of this file which is supplied in the *ies\observ_files* folder. This is because when I generated my version of *randobs.csv* by following the above steps, I had not already imposed limits on hydraulic conductivity in *plproc.dat*. Differences between the two versions are minor.

Note also that RANDOBS generates 500 realisations of noise-enhanced observations according to the above responses, whereas only 300 parameter realisations are adjusted by PESTPP-IES. The extra 200 observation realisations do no harm; PESTPP-IES simply ignores them.

6.5 Parameter Uncertainties

As an ensemble smoother, PESTPP-IES's task is to adjust random realisations of parameters that comprise samples of the prior parameter probability distribution until they become samples of the posterior parameter probability distribution. They are construed to sample the latter probability distribution when all adjusted parameter realisations allow model outputs to reproduce the calibration dataset to a level that is commensurate with measurement noise. A modeller can supply PESTPP-IES with prior parameter realisations him/herself. Alternatively, PESTPP-IES can generate these realisations. The latter option is adopted here.

When PESTPP-IES generates random parameter realisations itself, it centres them on initial parameter values supplied in the "parameter data" section of the PEST control file. It obtains parameter variances and covariances from a "parameter uncertainty file". This type of file is described in part 2 of the PEST manual. *Param.unc* is such a file. Refer to PEST documentation for specifications of this file type.

An inspection of *param.unc* reveals that it contains many COVARIANCE_MATRIX blocks. Each of these blocks cites a covariance matrix file that was previously used for regularisation purposes. Rows/columns of each covariance matrix are linked to parameters in the PEST control file using the "first_parameter/last_parameter" protocol.

File *param.unc* contains only one STANDARD_DEVIATION block. This block is actually surplus to needs because parameter *ppr_add1* is not adjusted by PESTPP-IES.

6.6 PESTPP-IES Control Variables

6.6.1 Overview

We now return to construction of the PEST control file that is used for the running of PESTPP-IES. Our starting point is *work4.pst* for, as discussed above, it retains observation weights that were previously used in calibration of the Milford model. Only a few changes need to be made to this file. These are now described. The final PEST control file is *milford1.pst*; this is provided in the *ies* working folder.

6.6.2 One-Sided Observations

Members of the PEST++ suite (of which PESTPP-IES is a member) use a different protocol for one sided observations than do members of the PEST_HP suite. To respect the PEST++ protocol, the name of observation group *<@river* must be changed to *g_river*. (The "*g_*" prefix indicates that a penalty is imposed if the modelled value is greater than the observed value.) The name of this observation group must be changed in the "observation groups" section of the PEST control file. It must also be changed in the "observation data" section of the PEST control file where observation *riv_reach2_1* is assigned to this group.

6.6.3 IES Control Variables

The following lines appear at the bottom of file *milford1.pst*. Lines which begin with "++" can be inserted anywhere in a PEST control file. The text string that follows these characters provides the value of a variable that controls the execution of programs that belong to the PEST++ suite. These can be supplied in any order.

```
++ies_num_reals(300)
++ies_enforce_bounds(true)
++ies_add_base(true)
++ies_autoadaloc(true)
++ies_subset_size(7)
++parcov(param.unc)
++ies_num_threads(4)
++ies_observation_ensemble(.\observ_files\randobs.csv)
```

The first of the above lines `ies_num_reals(300)` informs PESTPP-IES that it must use 300 parameter realisations, while the sixth line `parcov(param.unc)` informs PESTPP-IES that it must generate these realisations using parameter uncertainties recorded in file *param.unc*. (If a parameter uncertainty file is not supplied, PESTPP-IES assumes that every parameter is independently random, and that its standard deviation is a quarter of the difference between its upper and lower bounds.)

The last of the above lines `ies_observation_ensemble(.\observ_files\randobs.csv)` tells PESTPP-IES that it should not generate noise-enhanced realisations of observations. Instead, it should read these from file *random.csv*; preparation of this file is described above.

The control line `ies_enforce_bounds(true)` informs PESTPP-IES that all parameters in all realisations must respect parameter lower and upper bounds that are recorded in the "parameter data" section of the PEST control file.

`ies_autoadaloc(true)` instructs PESTPP-IES to use automatic adaptive localisation, while `ies_num_threads(4)` tells PESTPP-IES to use 4 computing threads to parallelize the numerical operations that localisation requires. (Note that the number of requested threads should not exceed the number of cores on your computer.)

The string `ies_subset_size(7)` instructs PESTPP-IES to use seven parameter realisations as a basis for Marquardt lambda testing during each iteration of the inversion process. The number of model runs that are committed to parameter upgrade evaluation using different Marquardt lambdas depends on lambda and line search settings. These can be provided to PESTPP-IES using pertinent control strings; in the present case, default settings are accepted.

Finally `ies_add_base(true)` informs PESTPP-IES that one of its random parameter realisations should not actually be random; instead, it should be comprised of initial parameter values cited in the "parameter data" section of the PEST control file. The observation realisation that is paired with this parameter realisation is comprised of observation values that are recorded in the "observation data" section of the PEST control file; they are not enhanced with a realisation of measurement noise.

As stated above, the above control strings can be supplied in any order. The writer admits that the ordering used in *milford1.pst* does not provide the best basis for explaining them.

6.7 Running PESTPP-IES

PESTPP-IES can commission model runs in serial or in parallel. In can operate in serial fashion if run using the command:

```
.\exec\pestpp-ies milford1
```

Like PEST_HP, PESTPP-IES uses the manager/agent concept to undertake model runs in parallel. However the agent is PESTPP-IES itself. The manager is run using the command:

```
.\exec\pestpp-ies milford1 /h :4004
```

An agent is run using the command:

```
.\exec\pestpp-ies milford1 /h %computername%:4004
```

if it is running on the same machine as the manager. Otherwise, replace `%computername%` with the hostname or IP address of the computer on which the manager is running.

6.8 PESTPP-IES Outputs

PESTPP-IES writes a suite of output files. These are described in its manual. Some of these files are provided in the *ies\results* folder. We mention only a few here.

milford1.phi.actual.csv tracks objective functions from iteration to iteration for all parameter realisations. Residuals used for calculating these "actual" objective functions are differences between realisation-specific model outcomes and observations recorded in the PEST control file (i.e. observations without measurement noise enhancement). Based on the contents of this file (which is easily examined in a spreadsheet) we adopt the ensemble calculated during iteration 3 as that which samples the posterior parameter probability distribution (after removing a number of poorly-performing realisations).

Parameter values calculated during this iteration are recorded in file *milford1.3.par.csv*.

The PEST suite provides programs that can extract individual realisations from a CSV file. CSV2PAR transfers parameter values that comprise a single realisation to a parameter value file. A PEST control file can then be populated with these parameter values using the PARREP utility (see above). CSV files can be converted to JCB files, wherein further parameter management options are available. If asked to do so, PESTPP-IES can record iteration-specific ensembles directly in a JCB file.

7. Calculating Probabilistic Contributing Areas

7.1 General

We now describe how to run the Milford model in order to calculate contributing areas (i.e. capture zones) for the Savage and Keyes extraction wells, and then how to run it repeatedly using different random parameter fields in order to calculate multiple contributing areas, and thereby contributing area probabilities, for these wells. As already stated, we use mod-PATH3DU for particle tracking. Repeated model runs using different parameter sets are supervised by PESTPP-SWP; this is a member of the PEST++ suite. (They could also have been supervised by PEST_HP, or even by a simple loop recorded in a batch file. However, with parameter realisations stored in a CSV file, PESTPP-SWP provides the easiest option.)

The working folder for the current section of this document is *conarea*. In addition to its usual subfolders, this folder has a subfolder named .*particles*. Most files in the *conarea* folder are unchanged from previous working folders. Where there are differences, these differences (and the reasons for them) are explained.

As usual, the .*exec* subfolder of the working folder contains executable programs that are required by the model. It should therefore be transferred along with the main folder to locations used by parallelisation agents. Note, however, that the .*exec* folder does not include the *mp3du.exe* and *writep3doutput.exe* executable files that are downloaded with mod-PATH3DU. You should download these yourself from the following website:

https://www.sspa.com/software/mod-path3du

7.2 Mod-PATH3DU

7.2.1 Overview

It is unlikely that capture zones evaluated using mod-PATH3DU (MP3DU) as a postprocessor for the Milford model would be any different from those evaluated using the USGS particle-tracking program MODPATH7. However the particle tracking algorithms employed by MP3DU on the one hand, and MODPATH7 on the other hand, have some important differences. These differences have little pertinence if a model grid is regular. However, in contrast to MODPATH7, MP3DU's algorithm can easily accommodate unstructured grids whose cells are not rectilinear in plan view. Simple extensions of its algorithm can accommodate extraction wells whose locations do not coincide with the centres of structured or unstructured cells.

Preparation of MP3DU input files was accomplished using Groundwater Vistas (GV). Some small modifications were made to some of these files prior to their use with the Milford model.

7.2.2 MP3DU Support Programs

The MP3DU suite is comprised of three executable programs. These are:

- *write3dgsf.exe*;
- *mp3du.exe*; and
- *writep3doutput.exe*.

WRITE3DGSF reads a MODFLOW-USG grid specification file. It writes another grid specification file in which elements of a model grid's geometry are recorded in a way that is better tuned to the needs of MP3DU. For the Milford model, this alternative grid specification file is named *milford1-new.gsf*; it is stored in the .*particles* subfolder. WRITE3DGSF needs to be run only once (unless changes are made to the geometry of the model grid).

MP3DU is the mod-PATH3DU executable program.

As it runs, MP3DU records particle trajectories in a large binary file. WRITEP3DOUTPUT reads this file. In accordance with user requests made through its input file, WRITEP3DOUTPUT transfers part or all of the contents of the binary MP3DU output file to one or a number of other files. In the present case we are interested only in its "ASCII Table" option. Programs belonging to the PEST Groundwater Utility suite can process the contents of this file in ways that are discussed below.

7.2.3 MP3DU Input Files

MP3DU reads a JSON input file. "JSON" stands for "JavaScript Object Notation". This is an ASCII file that holds structured data. For the Milford model, the MP3DU input file is *.\particles\milford1.json*. If invoked from the *conarea* working folder, MP3DU is run using the command:

```
.\exec\mp3du .\particles\milford1.json
```

Inspection of *milford1.json* reveals the IFACE settings that are ascribed to different MFUSG boundary types. The "6" ascribed to the RCH, EVT, DRN and RIV boundary types informs MP3DU that if a particle ends its journey in a cell which is occupied by any of these types of boundary, then its journey ends at the top face of the model cell.

.\particles\milford1.json also informs MP3DU of the "name file" that it must read. This file is named *milford1-mpath.nam*; it resides in the *conarea* folder. This MP3DU name file is identical to the MFUSG name file *milford1.nam* except for the fact that it cites an extra input file, this being *.\particles\milford1.p3d.* An inspection of *.\particles\milford1.p3d* reveals that this file contains a suite of model-layer-specific datasets. Values for all of these sets are uniform in each model layer; hence they are supplied using the CONSTANT specifier.

.\particles\milford1.json also informs MP3DU that it should read particle starting locations from a shapefile named *milford1-ptstart.shp*. This file (and its associated files) was written by GV. When designating particle starting locations in GV, I used its polygon-select tool to define an area that is slightly larger than the active part of the Milford model grid. I asked GV to place a particle at the top of each model cell within this polygon. Particles that are placed in inactive cells go nowhere. By definition, a particle whose path terminates at either the Savage or Keyes public supply well begins its journey at a place that contributes water to that well; it is therefore in the contributing area of the well.

By asking GV to place particles at the top of each cell, each particle is endowed with an initial ZLOC value of 1.0. (ZLOC specifies the local *z* coordinate of a particle.) In retrospect, it would have been better to specify initial ZLOC values of 0.95. Sometimes, where multiple boundary conditions are allocated to the same groundwater model cell, the version of MP3DU that was used for work that is documented herein can get stuck in an infinite loop. This happened for a few of the random parameter fields that were used in the procedure described below for undertaking probabilistic contributing area analysis. It made automation of this process a little more difficult than it should have been. It also created computational inconveniences, as MP3DU's binary output file becomes inordinately large if MP3DU gets stuck in an infinite loop. (Fortunately, because I was using a 4GB RAM disk; MP3DU ceased execution when the size of this file reached 4GB.) These problems could have been prevented by specifying an initial ZLOC value of 0.95 for all particles. This was done when undertaking probabilistic contributing area analysis using the single layer version of the Milford model that is described in the worked example report.

(If you wish to follow the steps that are presented below for the three layer Milford model, it is recommended that you open *milford1-ptstart.shp* in a GIS package such as QGIS, and alter all ZLOC values from 1.0 to 0.95.)

7.3 Calculation and Display of Contributing Areas

7.3.1 Particle Path File

As stated above, MP3DU records a binary file that contains details of particle tracks that it calculates. The WRITEP3DOUTPUT utility supplied with MP3DU can write an "ASCII table" file based on the contents of this binary file. This ASCII file is not unlike particle track files that are written by the USGS MODPATH family of programs. In the present instance, this file is named *milford1.ptl*. It is very large in size as it records the tracks of nearly 34000 particles. (Because of its size, you will need to run MP3DU and WRITEP3DOUTPUT yourself to obtain a copy of this file.)

Columns in this file are as follows:

- PID: the particle identifier. These are sequential and start at 1.
- CELLID: the cell in which the particle presently resides. A cell is identified by its MFUSG node number.
- PTIME: the current simulation time.
- PX: the particle's current *x* coordinate.
- PY: the particle's current *y* coordinate.
- PZGLO: the particle's current global *z* coordinate.
- PZLOC: the particle's current local (i.e. cell-based) *z* coordinate.
- PVX_ADV: the particle's current advective velocity in the *x* direction.
- PVY_ADV: the particle's current advective velocity in the *y* direction.
- PVZ_ADV: the particle's current advective velocity in the *z* direction.
- PLAYER: the layer number in which the particle currently resides.
- PTSTART: the time at which the particle commenced its journey.

7.3.2 Determining Contributing Area

Program MP3DUFBE is a member of the PEST Groundwater Utility suite. It reads an "ASCII table file" (i.e. a particle path file) written by WRITEP3DOUTPUT. It also reads a user-prepared "endpoint filter file"; this file records MFUSG model cells for which a user would like to know travel details of particles that end in these cells. MP3DUFBE writes two files. The first of these files records the starting cell of each particle featured in a particle path file that terminates its journey in one of the user-specified endpoint cells. It also records the total travel time of each such particle. We refer to this MP3DUFBE output file as a "startpoint file".

MP3DUFBE also records an "abridged MP3DU output file". This is a particle path file that features the paths of only those particles whose journeys end in user-specified endpoint cells.

7.3.3 Displaying Particle Data

The startpoint file produced by MP3DUFBE is easily imported into a GIS platform for direct display of contributing area. Particle starting points can be plotted on a map, and coloured according to total particle travel time. Figures that are provided in Appendices B and C of the Milford worked example report display contributing areas in this way.

At the time of writing, the PEST Groundwater Utility suite provides two programs which facilitate display of particle paths.

The abridged particle path file written by MP3DUFBE (or the full particle path file written by WRITEP3DOUTPUT) can be read by the MP3DU2VTK utility. This program writes a VTK file which can be imported into a package such as PARAVIEW in order to display particle tracks in three dimensions. Figures 5.4 and 5.5 from the Milford worked example report are screenshots of PARAVIEW displays that are based on a VTK file written by MP3DU2VTK.

The abridged particle path file written by MP3DUFBE (or the full particle path file written by WRITEP3DOUTPUT) can be read by the MP3DU2MIF utility. This produces a MIF/MID file pair. ("MIF" stands for "Mapinfo Interchange Format".) MIF/MID files can be imported into a GIS. The Milford worked example report provides no examples of particle paths displayed in a GIS. Some pictures are shown below. In these pictures, particle paths are coloured according to the time since the particle commenced its journey.

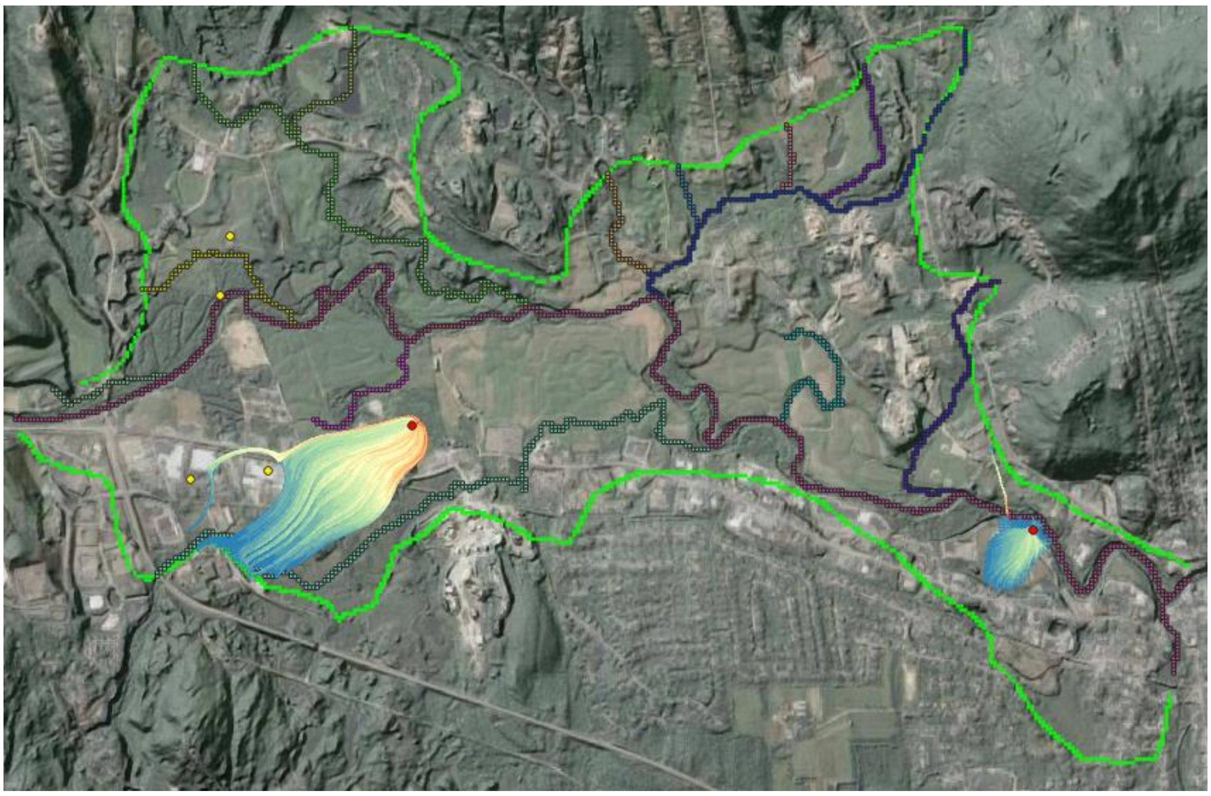

Figure 7.1. Tracks of particles captured by the Savage and Keyes wells coloured according to residence time.

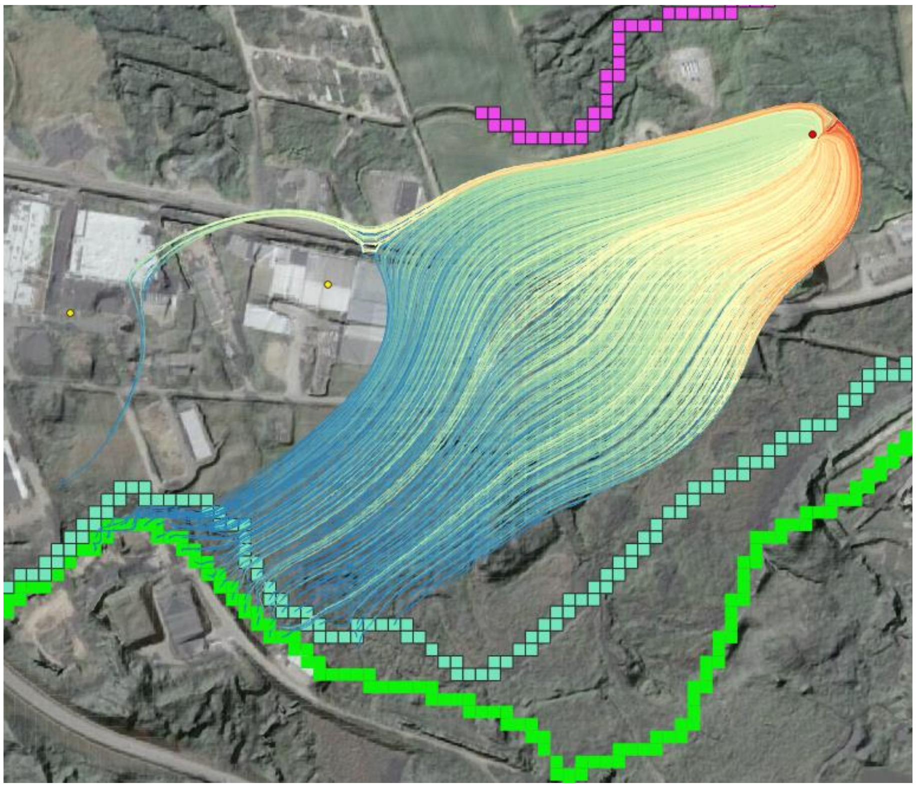

Figure 7.2. Closeup view of part of Figure 7.1.

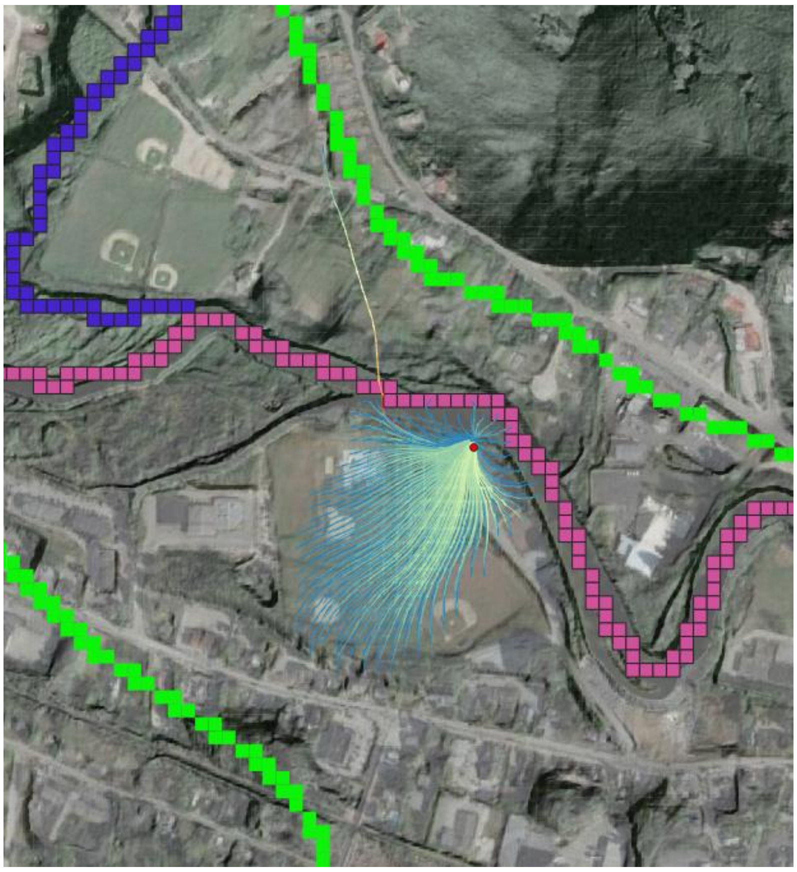

Figure 7.3. Closeup view of part of Figure 7.1.

7.4 Probabilistic Contributing Areas

7.4.1 Overview

Programs that are described above enable definition and display of the area that contributes water to an extraction well. Calculation and display of probabilistic contributing area is more complex than this. It requires that the contributing area be calculated many times using many different samples of the posterior parameter probability distribution. From the outcomes of all of these model runs, statistics can be accumulated to define contributing area probabilities. This procedure is now described.

As usual, a PEST dataset is used to specify relationships between model parameters and model inputs. The PESTPP-SWP program from the PEST++ suite can read such a file. It can commission many model runs based on many different parameter sets. On each occasion that it runs the model, PESTPP-SWP reads a new set of parameters from a CSV file.

When undertaking probabilistic contributing area analysis, definition of “the model” must change from that which was employed when PEST_HP and PESTPP-IES were used for history-matching. The model must now be comprised not just of MFUSG, but also of MP3DU. As is discussed above, cells in which particles that are captured by a water supply well originate collectively define the contributing area for that well. The number of times that a cell features in a contributing area, when that area is calculated repeatedly using many different random parameter fields, defines its probability of lying within the real contributing area for that well.

Details of this process are now provided.

7.4.2 Modifications to the PEST Control File

File *milford1.pst* residing in the *conarea* folder is identical to file *milford1.pst* residing in the *ies* folder, except for two things:

- The NOPTMAX variable is set to 50 (this is arbitrary).
- A control statement for PESTPP-SWP has been added to this file.

The PESTPP-SWP control statement is as follows.

```
++sweep_parameter_csv_file(.\results\milford1.3.par.csv)
```

This statement directs PESTPP-SWP to undertake many model runs. In undertaking these runs, it draws parameter sets from successive rows of the CSV file *.\results\milford1.3.par.csv*.

7.4.3 Modifications to MFUSG Input Files

When calibrating the Milford model, pumping from the Savage and Keyes wells was disabled. When evaluating contributing areas to these wells, pumping must be turned on. This is done in a new *milford1.wel* file; see the end of this file where extraction from CLN nodes is listed.

When calibrating the Milford model, recording of cell-by-cell flow terms in file *milford1.cbb* was switched off for a number of MFUSG packages. It must be switched on for all packages in circumstances where particle tracking is undertaken.

7.4.4 Modifications to the Model Batch File

The model batch file *model.bat* now includes an extra line. This line calls another batch file named *run_mp3du.bat*.

File *run_mp3du.bat* begins with a set of deletion commands. As is explained above, this is a quality assurance measure. It prevents an old file from being mistaken for a new one if any program in the processing sequence fails to run.

Next *run_mp3du.bat* runs the MP3DU particle tracker. This is followed by WRITEP3DOUTPUT. Next the MP3DUFBE utility is used to record a particle startpoint file. Responses to MP3DUFBE's keyboard prompts reside in file *.\particles\mp3dufbe.in*. As is apparent from this file, the model cells of interest as far as particle capture is concerned must be read from file *.\particles\endcells.dat*. These are GWF cells to which connected linear networks representing the Savage and Keyes extraction wells are connected. MP3DUFBE records the starting points of captured particles in file *.\particles\captured.txt*.

Next *run_mp3du.bat* runs the MP3DUACCUM utility. This is another member of the Groundwater Utility suite. Keyboard responses to MP3DUACCUM's prompts are provided in file *.\particles\mp3duaccum.in*.

As the name suggests, MP3DUACCUM accumulates the outcomes of many MP3DU runs. It reads the startpoint file produced by MP3DUFBE; as stated above, this file is *.\particles\captured.txt*. It also reads a node data table file in which cell counts are being accumulated. In the present case this file is named *.\particles\cap_his.ndt*. MP3DUACCUM augments cell contributing area counts in this file.

If MP3DUACCUM does not find the node data table file that it is directed to read and update, then it creates one itself; all cell counts are set to zero in this file. It is important, therefore, that if this file exists before PESTPP-SWP is run, that it be deleted. As will be demonstrated in the next section, data that resides in a node data table file is easily imported into a GIS for display, or into PARAVIEW for 3D visualisation.

The contributing area counts in file *cap_his.ndt* are easily converted to contributing area probabilities. Each count must simply be divided by the total number of realisations that are used in particle count accumulation.

7.4.5 Running PESTPP-SWP

Like other members of the PEST++ suite, PESTPP-SWP can undertake model runs in serial or in parallel. In the present case they are undertaken in serial. See file *run_sweep.bat*.

8 Visualisation and Display

8.1 General

We finish this document by mentioning a few utility programs that can expedite visualisation and display of MFUSG input and output datasets. These (and other) programs are listed in Appendix A of the Milford worked example report. For full details see Part C of the manual of the PEST Groundwater Utilities suite.

Note that programs which facilitate visualisation and display of MP3DU-calculated particle tracks and MP3DUFBE-identified starting points are discussed in the previous section of this document.

8.2 Data Exchange with a GIS

If MFUSG input datasets and/or MFUSG output datasets are translated to node data table format, then they can be imported into a GIS. A MFUSG grid specification file is also required.

Program USGNDTF2MIF reads a MFUSG grid specification file as well as a node data table file. The latter can contain multiple columns of real and/or integer data. Numbers residing in one or more of these columns can be used as a "filter" so that only parts of the grid are imported into a GIS (for example, only active cells). After data column selection is performed by the user in response to USGNDTF2MIF's prompts, it then writes a MIF/MID file pair for a single, user-specified, layer of the model grid. This pair of files is easily imported into a GIS. Once imported, data contained therein can be saved as a shapefile for faster access next time. These data can also be edited and saved in tabular format; with a little manipulation, they can then be used by the model in place of its original data. If desired, the GIS layer can be endowed with a coordinate reference system within the GIS itself.

The PEST Groundwater Utility suite provides a number of mechanisms through which model input/output datasets can be recorded in node data table format. Some of these have already been discussed. In particular:

- USGPROP2TAB1 and USGPROP2TAB2 can extract cell properties and activities from MFUSG input files.
- TAB2NDTF can reformulate a table that supplies data for only some cells of a model grid as a node data table in which a default value is supplied for model cells that are not cited in the original table.
- USGBIN2TAB_H can extract system states (heads and concentrations) from a MFUSG-written binary system state file.

If desired, data from many sources can be placed into a single node data table file using the column cut and paste functionality provided by high-end text editors. These data can then be imported into a GIS platform in a single operation.

8.3 Data Exchange with PARAVIEW

Programs USG2VTK and USG2VTK1 read a MFUSG grid specification file. Optionally, they also read a node data table file. Geographical and cell-based data that resides in these files can then be recorded in a VTK file. A model grid, together with values assigned to model cells, can then be displayed and coloured in three dimensions. See below.

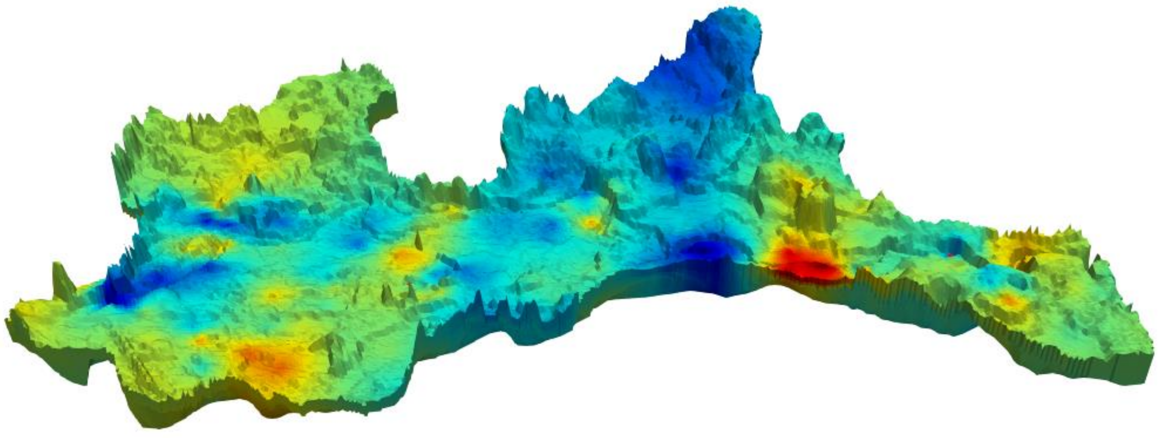

Figure 8.1 The Milford model grid shaded according to log of hydraulic conductivity.

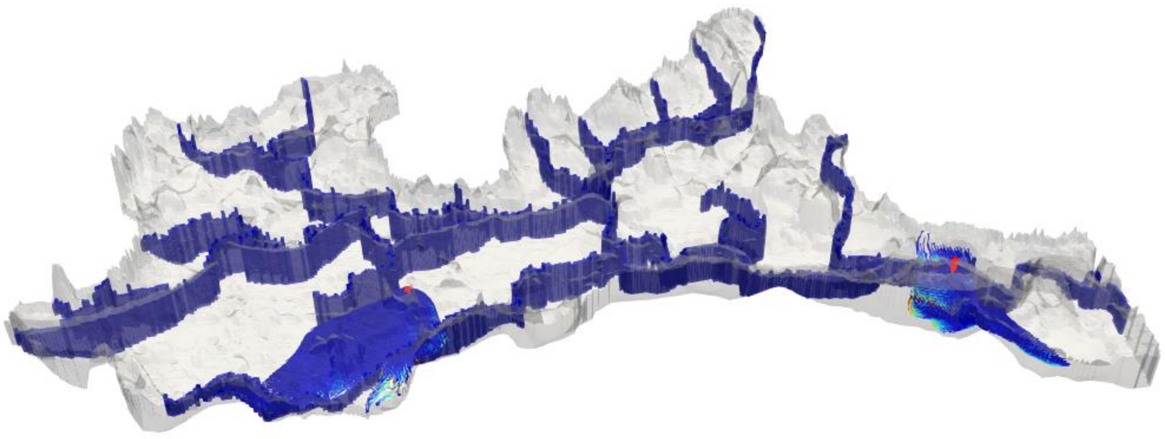

Figure 8.2 The Milford model grid, river cells and tracks of captured particles.

The PEST Groundwater Utility suite includes a program named USGMODGSF. This program modifies a MODFLOW-USG grid specification file so that the tops and bottoms of all model cells are displayed as flat; cell top and bottom elevations are extracted from a MFUSG discretisation file. If USG2VTK or USGVTK1 is asked to construct a VTK file on the basis of this grid specification file, the grid appears "blocky" when visualised. This "blockiness" allows the user to see the grid "as the model sees it" – and also as particles see it. This may allow a user to better understand visualised tracks of particles, particularly at places where they appear to suffer vertical displacements as they cross cell boundaries.