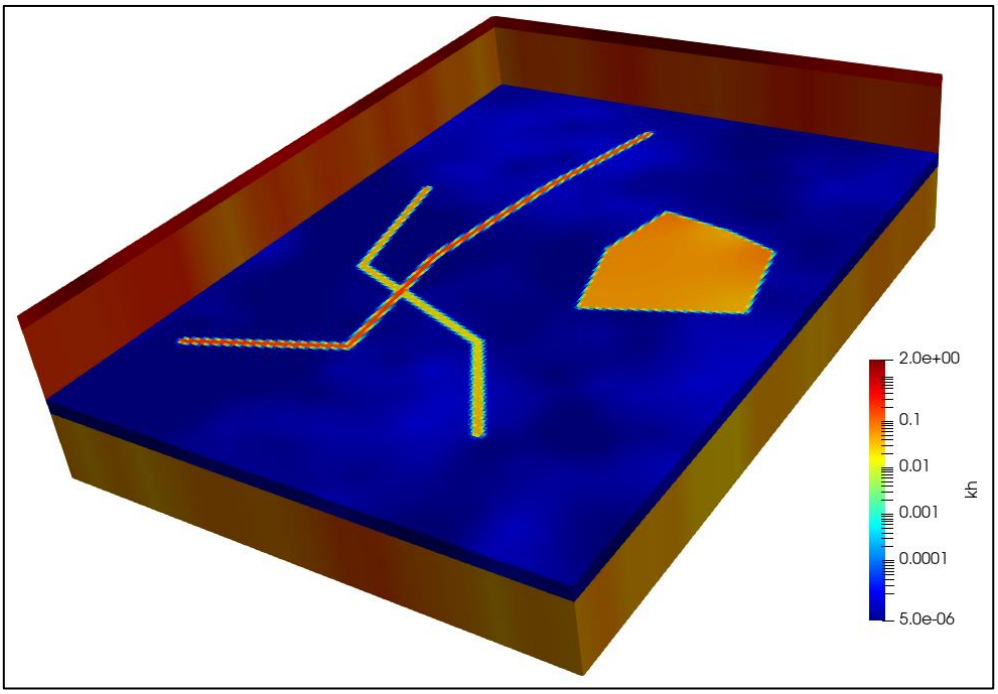
Tutorial

Moveable Structures
and their
Hydraulic and Positional Parameters



John Doherty

May, 2023

gmdsi
Groundwater Modelling
Decision Support Initiative

Table of Contents

1. Introduction

This tutorial shows you how to work with structural overlay parameters. These are supported by PLPROC, a parameter preprocessor that comes with PEST. Some other facets of PLPROC functionality (as well as that of some members of the PEST Groundwater Utility Suite) are also demonstrated in this tutorial.

As the name suggests, structural overlay parameters can be overlain on parameter fields that have already been assigned to a model. Hence they can represent post-depositional features such as faults. They can also represent the effects of local unconformities that create holes in aquitards.

Structural overlay features can be polylinear, or they can be polygonal. Their shapes are defined by their vertices. A hydraulic property (such as hydraulic conductivity) can be assigned to each of its vertices. For polylinear features this property is linearly interpolated between vertices; for polygonal features it is spatially interpolated within a polygon that is bounded by these vertices using the inverse-power-of-distance method.

Structural overlay features are moveable. Their vertices can glide along "sliders". Vertex locations along their sliders can be adjusted by PEST at the same time as vertex hydraulic properties are adjusted. A second level of structural feature positional adjustment is also possible; all of the sliders that determine the shape of a particular structural feature can glide along their own slider as a single unit. Therefore, as will be demonstrated herein, the shape and properties of an aquitard hole can be adjusted at the same time as the entire hole is moved along a dedicated slider.

Caution must be exercised when formulating an inverse problem in which geobody locations are adjustable parameters. Ideally, the relationship between all model outputs and any model parameter should be continuous. This requires that the hydraulic properties that are assigned to polylinear and polygonal features such as faults and aquitard holes "blend" into those of the material that hosts them rather than changing sharply at their boundaries. The "blending distance" is set by the user; it can be large or small. While hydraulic property blending may seem a little artificial, it must be kept in context. Numerical simulation is replete with artificialities; hydraulic property blending is a minor artificiality compared with most that afflict groundwater modelling. Furthermore, spatial blending of hydraulic properties enables simulator-based data assimilation; this is a fundamental requirement of decision-support modelling.

This tutorial is based on a simple, multi-layer, MODFLOW 6 model. We will not calibrate this model. However we will build a PEST input dataset that enables its parameterisation. Parameters will include:

- hydraulic conductivities at pilot points (these comprise the background model parameterization scheme);
- hydraulic conductivites at the vertices of two faults and an aquitard hole;
- the locations of the vertices that define these structural features;
- the position of the aquitard hole along a single, extended slider.

Probability distributions will be ascribed to all of these parameter types. Using these probability distributions, random parameter realisations will be drawn. Model hydraulic properties that are calculated from these parameters will be viewed as three-dimensional plots of the model domain.

2. Files

Unzip the tutorial file. You will find a base working folder, as well as three subfolders named:

- pictures,
- results,
- documentation.

The *documentation* folder contains the document that you are reading now. The base folder contains the executable programs and files and are required for the tutorial. This, and to some extent the *pictures* subfolder, are our working folders. Start this tutorial by opening a command line window in the base working folder.

The *results* subfolder contains the same files as the base folder (except for executable programs), as well as files that are produced in the course of undertaking this tutorial.

3. The Model

The model domain is rectangular. The grid is structured; it has a 4 layers, 200 rows and 150 columns. The model's name file is *model.nam*. It is depicted in Figure 3.1.

```
BEGIN options
  NEWTON
END options

BEGIN packages
  DIS6  model.dis  dis
  IC6   model.ic   ic
  NPF6  model.npf  npf
  OC6   model.oc   oc
  CHD6  model.chd  chd
END packages
```

Figure 3.1. The model name file *model.nam*.

The model's geometry is simple; it has no inactive cells. Layer 1 represents weathered material of moderate to high permeability. Layers 2 and 4 represent aquifers, while layer 3 is an aquitard. The tops and bottoms of all model layers are horizontal. The structured discretisation file *model.dis* specifies this simple geometry.

Only a single boundary condition is operative, this being the CHD (i.e. constant head) condition. Inspect file *model.chd*. You will see that a head of 235 m is assigned to the northernmost row of cells in layers 2 and 4, while a head of 225 m is assigned to the southernmost row of cells in these same layers. Water therefore flows from north to south. However this is of secondary concern to the current tutorial. We are interested only in the model's parameterisation.

Nevertheless, we will run the model, for there is one model output file that is of particular interest to us. This is the binary grid file (or "GRB file" for short) that is recorded by MODFLOW 6. PLPROC reads this file in order to obtain details of the MODFLOW 6 model grid. It needs to know these details so that it can interpolate from pilot points and other spatial parameterisation devices (including structural overlay parameters) to grid cell centres.

In this tutorial, we are interested only in parameterization of hydraulic conductivity. (The model does not require storage parameters as it runs in steady state.) An inspection of file *model.npf* (the input file for the Node Property Flow package) reveals that:

- K33 is not mentioned in this file; therefore vertical hydraulic conductivities are equal to horizontal hydraulic conductivities;
- Hydraulic conductivities for all layers are read from a single file named *k.txt*.

Inspect file *k.txt*. Hydraulic conductivities for all model cells are recorded one to a line. All cells in layer 1 are assigned a hydraulic conductivity of 1.0 m/day. Hydraulic conductivities in layers 2, 3 and 4 are uniformly 0.1 m/day, 1.0E-5 m/day and 0.05 m/day respectively.

Run the model by typing:

```
mf6
```

at the screen prompt. The model runs in a little over a second. You will see that it writes a file named *model.dis.grb*. This is the binary grid file.

4. Looking at the Model

The PEST Groundwater Utility Suite provides a number of programs which can help you to visualise a model in different ways, and to import/export model data from/to a geographical information system (GIS). We will make some pictures of the model using SURFER and PARAVIEW. (Do not worry if these programs are not installed on your machine; they are not an essential part of this tutorial.) Files which can be imported into these packages will be recorded in the *pictures* subfolder.

First, run the MF62GIS utility to create a SURFER BLN file of the model grid. Type "MF62GIS" at the screen prompt while situated in the base working folder; then respond to its prompts in the manner that is shown below.

```
Program MF62GIS writes a BLN file and MIF/MID files for a MODFLOW6 model.

Enter name of MODFLOW6 binary grid file: model.dis.grb
- file model.dis.grb read ok.

Enter layer number of interest: 1

Enter name for BLN file (<Enter> if none): .\pictures\grid.bln

Enter filename base for MIF/MID files (<Enter> if none): <Enter>
- file .\pictures\grid.bln written ok.
```

If the BLN file is imported into SURFER as a "base layer", you will see that the model grid looks like this (unsurprisingly).

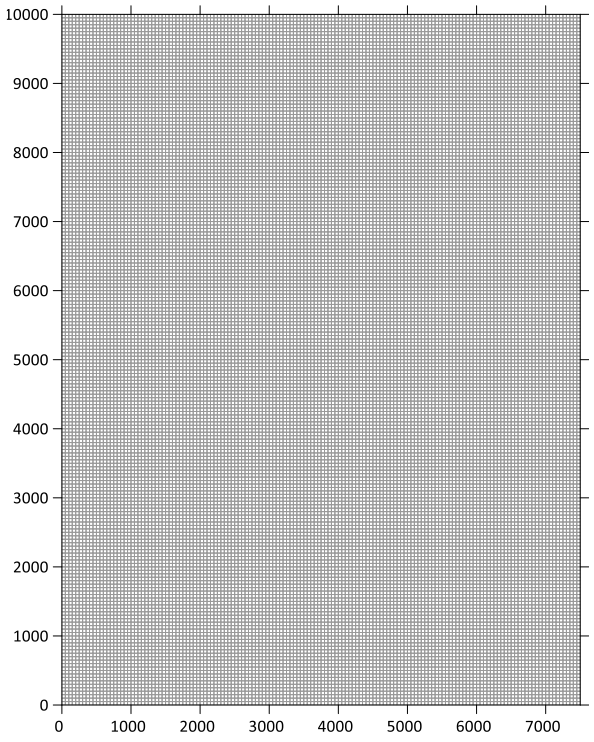


Figure 4.1. The model grid.

PARAVIEW can provide a three dimensional view of the model domain. It can also be used to display model boundary conditions. Run the MF62VTK1 utility from the working folder by typing its name at the screen prompt while situated in this folder. Then respond to its prompts as follows.

```
Program MF62VTK1 writes a "legacy" VTK file based on a MODFLOW6 binary grid
  output file and, optionally, associated node data.

Enter name of MODFLOW6 binary grid file: model.dis.grb
```

```
 - file model.dis.grb read ok.

 Enter name for VTK output file: .\pictures\grid.vtk

 Record scalar data in VTK file?  [y/n]: y

 Enter model input file #1 (<Enter> if no more): model.chd
   Enter starting text for data table: BEGIN PERIOD
   Enter finishing text for data table: END PERIOD
   Number of columns in table = 4
   In what column do cellid's begin? 1

   Enter label for data in column #4 (<Enter> if ignore): CHD_HEAD
   Is this integer or real data? [i/r]: r
   Enter value for missing cells: 0.0
   Add or replace values for duplicated cells [a/r]: r
 - 600 lines of data read from file model.chd.

 Enter model input file #2 (<Enter> if no more): <Enter>

 - file grid.vtk written ok.
```

File *grid.vtk* (recorded in the *pictures* subfolder) can be imported into PARAVIEW. Figure 4.2 shows the model grid (looking from the southeast), with the model domain coloured according to layer number. Figure 4.3 is a cutaway view of the model grid with CHD boundaries coloured red.

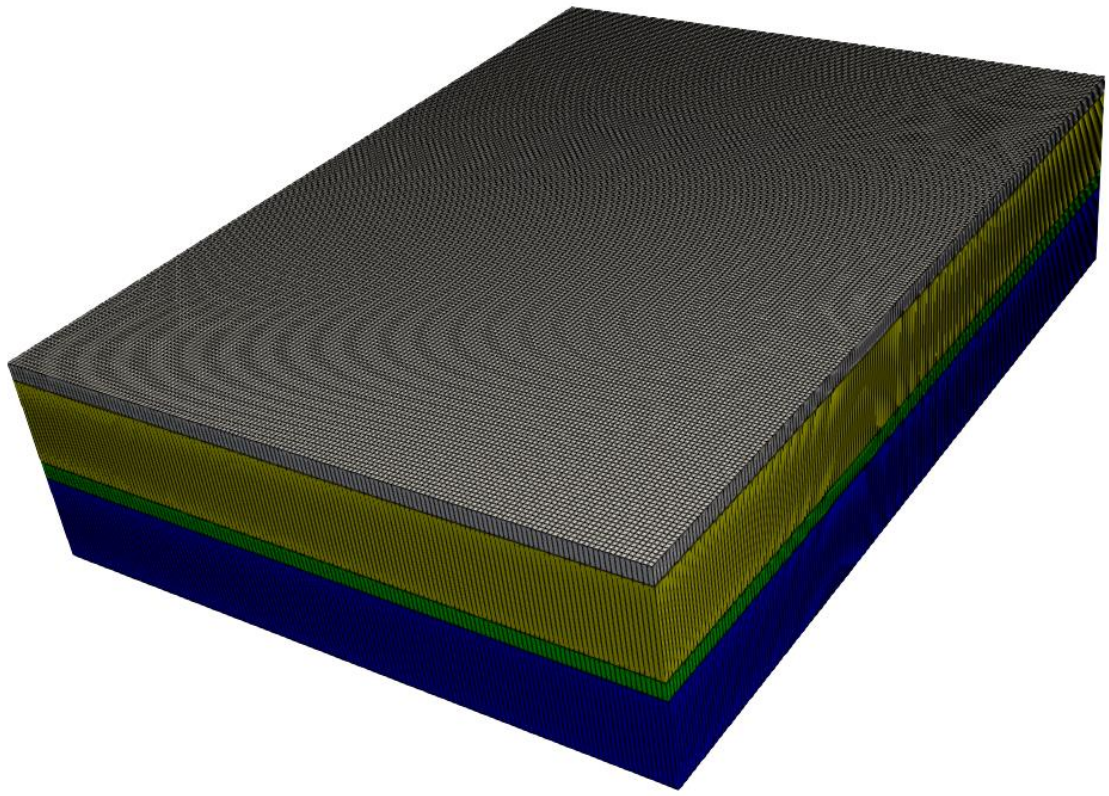


Figure 4.2. The model domain, coloured according to layer. The vertical exaggeration is 10.

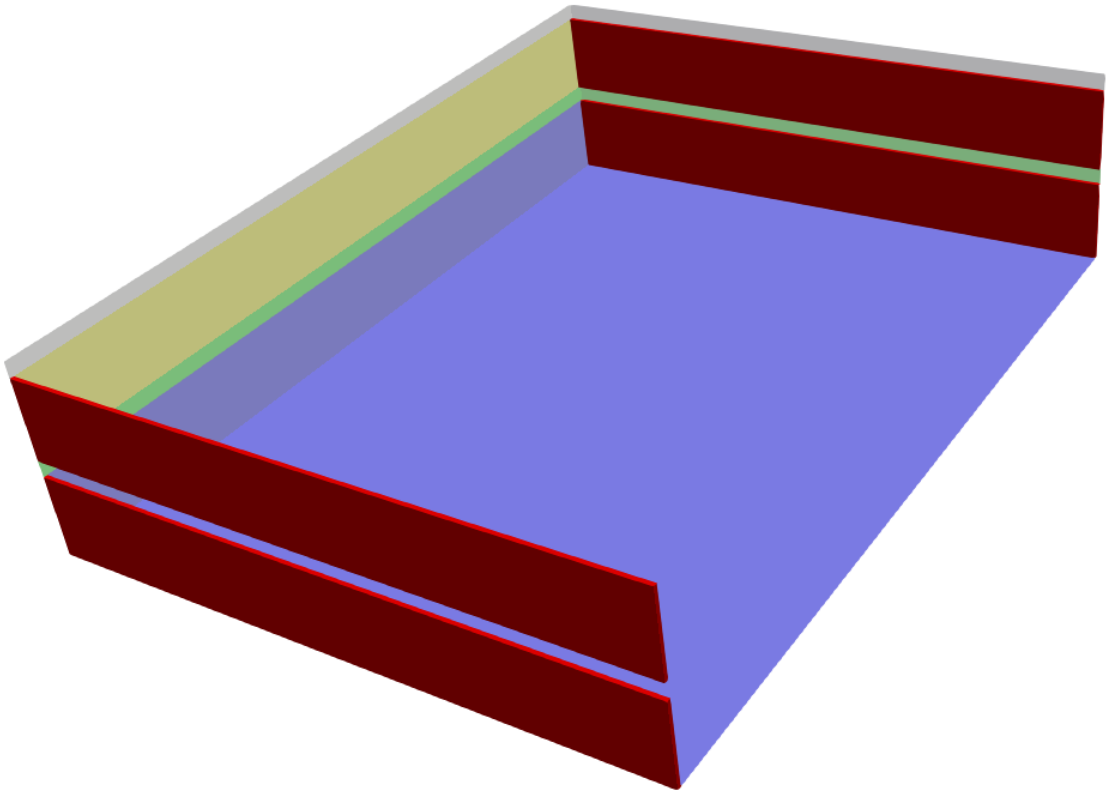


Figure 4.3. Model CHD boundaries are shown in red while model cells are coloured according to layer.

Before finishing this section, another display-related utility program will be demonstrated. We will use it to plot model-calculated heads in PARAVIEW. First these must be extracted from the MODFLOW 6 binary heads file *model.hds* in which they reside using the MF6DEP2CSV utility. The contents of the CSV file which MF6DEP2CSV produces will then be recorded in a VTK file.

Run MF6DEP2CSV, responding to its prompts as follows.

```
Program MF6DEP2CSV records MODFLOW6-calculated system states in CSV format.

Enter name of MODFLOW6 binary grid file: model.dis.grb
- file model.dis.grb read ok.

Enter binary MF6-generated dependent variable file: model.hds

Enter name for CSV output file: heads.csv
Record "HEAD" data for all model layers or just one? [a/o]: a
Record "HEAD" data for all model output times or just one? [a/o]: a
- pre-reading file model.hds...
- reading file model.hds...
- file model.hds read ok.
- writing file model.csv...
- file model.csv written ok.
```

Now run MF62VTK2:

```
Program MF62VTK2 writes a "legacy" VTK file based on a MODFLOW6 binary grid
  output file and an MF6DEP2CSV-written CSV file.

Enter name of MODFLOW6 binary grid file: model.dis.grb
- file model.dis.grb read ok.

Enter name of MF6DEP2CSV-produced CSV file to read: heads.csv

Enter name for VTK output file: .\pictures\model_heads.vtk
- reading file heads.csv...
```

```
 - file heads.csv read ok.
 - file .\pictures\model_heads.vtk written ok.
```

If PARAVIEW is installed on your machine, import file *\pictures\model_heads.vtk* to produce a picture like that shown below (in which colours pertain to model-calculated heads).

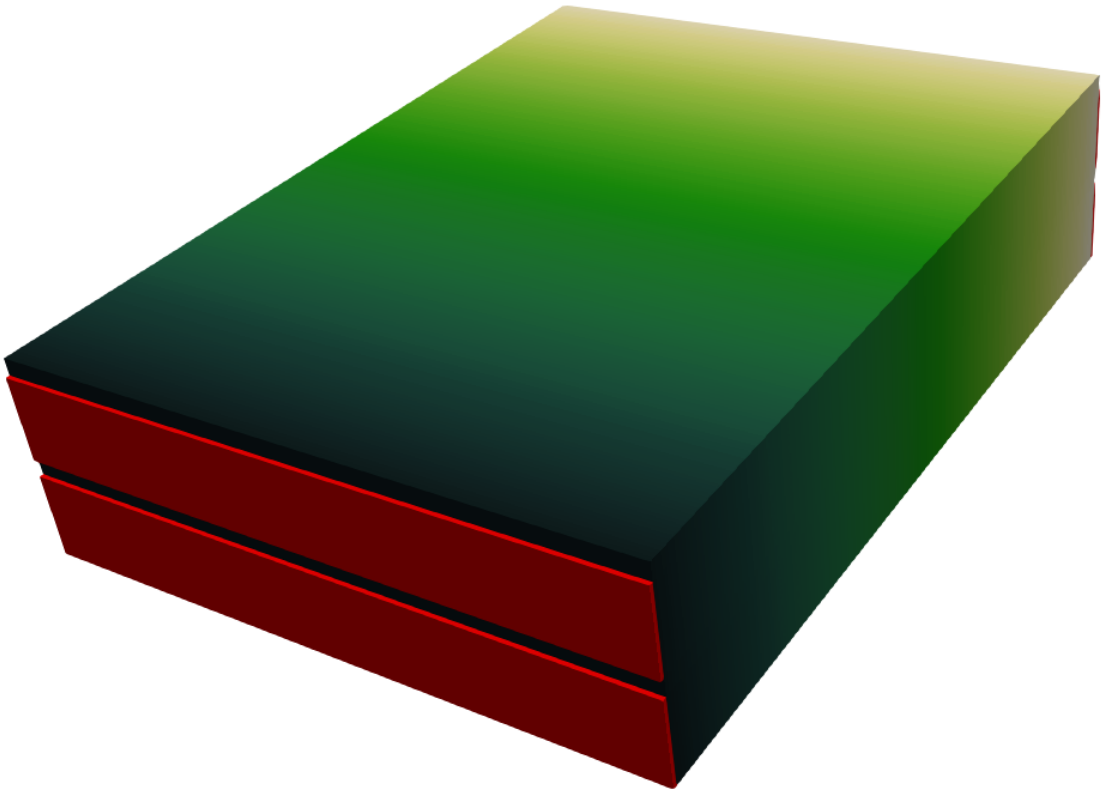


Figure 4.4. Model-calculated heads and CHD boundary conditions. Deeper green signifies a lower head.

5. Pilot Points

5.1 Pilot Point Locations

Background parameterization of hydraulic conductivity for all model layers employs pilot points. Inspect file *pp.dat*. This provides pilot point locations, together with four values of hydraulic conductivity that are associated with each of them. These values pertain to the four layers of the model. A PEST template file that corresponds to this pilot points file has also been prepared. See file *pp.tpl*. Pilot point parameters are named in this template file; the name of each parameter includes the layer to which it belongs.

Pilot points are placed 300 m apart on a regular grid. Based on the contents of file *pp.dat*, they can be superimposed on the model grid using SURFER. See Figure 5.1.

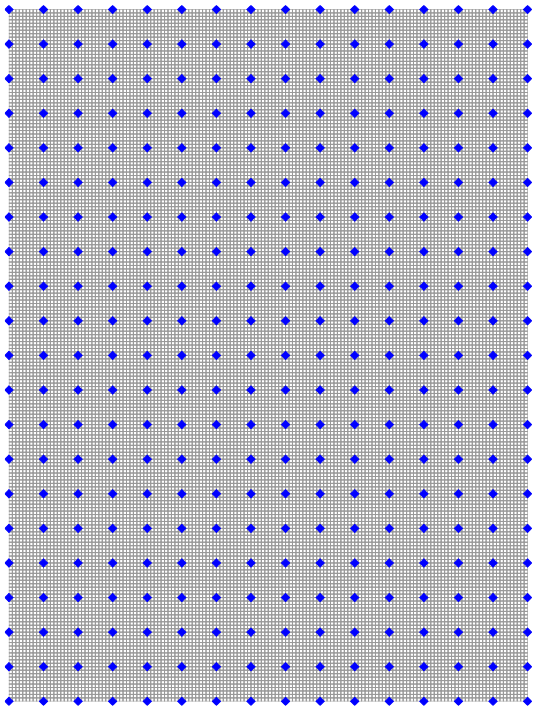


Figure 5.1. Pilot point locations.

5.2 PLPROC Script

File *plproc1.in* is a PLPROC script that interpolates hydraulic property values that are ascribed to pilot points to cells within the four layers of the MODFLOW 6 model grid. We will describe the contents of this PLPROC input file line by line. Refer to the PLPROC manual for more details of PLPROC functionality as it pertains to pilot points.

First PLPROC is directed to read the binary grid file that was written by MODFLOW 6. From this file it obtains geometric specifications of the model grid. (It is incumbent on a modeller to ensure that correct grid origin coordinates, and the correct grid rotation angle, are provided in the model's DIS file, in this case *model.dis*. If this is not done, then pilot point coordinates cannot be properly related to model grid coordinates.) The PLPROC function call follows. This call creates a three-dimensional CLIST named *cl_mf6*. Each element of this CLIST pertains to an individual cell of the model grid. A complementary SLIST named *layer* records the layer number to which each model cell belongs.

```
# -- Read the MODFLOW 6 GRB file.

cl_mf6 = read_mf6_grid_specs(file=model.dis.grb,        &
                             dimensions=3,              &
                             slist_layernum = layer)
```

PLPROC is next instructed to read file *pp.dat*. A two-dimensional pilot points CLIST named *cl_pp* is created, together with four complementary PLISTs. Each of these PLISTs hosts pilot-point-based hydraulic conductivity values for a discrete model layer; these values will shortly be interpolated to cells within the corresponding layer of the model grid.

```
# -- Read the pilot points file to define a pilot points CLIST, and
#    to obtain K values at all pilot points in all layers.

cl_pp = read_list_file(file='pp.dat',              &
                       skiplines=1,                &
                       dimensions=2,               &
                       id_type='character',        &
                       plist='pp_k1';column=4,     &
                       plist='pp_k2';column=5,     &
                       plist='pp_k3';column=6,     &
                       plist='pp_k4';column=7)
```

Kriging factors are next calculated using PLPROC's *calc_kriging_factors_auto_2d()* function. This function is called four times, one for each layer. Once these interpolation factors have been calculated, these lines of the PLPROC script can be commented out, for kriging factors only need to be re-calculated if the locations of pilot points change, or if the locations of grid cell centres change.

```
# -- Calculate kriging factors from pilot points to the model grid.

calc_kriging_factors_auto_2d(target_clist=cl_mf6;select=(layer==1),      &
                             source_clist=cl_pp,                         &
                             file=factors_lay1.dat;format=binary)

calc_kriging_factors_auto_2d(target_clist=cl_mf6;select=(layer==2),      &
                             source_clist=cl_pp,                         &
                             file=factors_lay2.dat;format=binary)

calc_kriging_factors_auto_2d(target_clist=cl_mf6;select=(layer==3),      &
                             source_clist=cl_pp,                         &
                             file=factors_lay3.dat;format=binary)

calc_kriging_factors_auto_2d(target_clist=cl_mf6;select=(layer==4),      &
                             source_clist=cl_pp,                         &
                             file=factors_lay4.dat;format=binary)
```

Next a new PLIST is created. This is a child PLIST of the *cl_mf6* CLIST; hence it represents model grid cells. This PLIST will eventually host K values that are interpolated from pilot points.

```
# -- An MF6 K PLIST is now defined.

mf6_k=new_plist(reference_clist=cl_mf6,value=1.0)
```

Now that the *mf6_k* PLIST has been created, hydraulic conductivity values that are ascribed to pilot points can be spatially interpolated to the MODFLOW 6 grid in order to populate the PLIST. This is done for each model layer using each of the previously-calculated set of interpolation factors.

```
# -- Interpolation to the MF6 grid is now carried out.

mf6_k=pp_k1.krige_using_file(file='factors_lay1.dat';form='binary',     &
                             transform='log')

mf6_k=pp_k2.krige_using_file(file='factors_lay2.dat';form='binary',     &
                             transform='log')

mf6_k=pp_k3.krige_using_file(file='factors_lay3.dat';form='binary',     &
                             transform='log')

mf6_k=pp_k4.krige_using_file(file='factors_lay4.dat';form='binary',     &
                             transform='log')
```

Hydraulic conductivity values can now be delivered to the MODFLOW 6 model. Recall that MODFLOW 6 reads these values from a file named *k.txt*. This file is now written. Notice how PLPROC uses a template file of *k.txt* named *k.txt.tpl* to write file *k.txt*.

```
# -- The file holding the MODFLOW K array is written.

write_model_input_file(template_file='k.txt.tpl',      &
                       model_input_file= 'k.txt')
```

The contents of file *k.txt.tpl* are shown below. This contains an embedded PLPROC function which informs PLPROC how it should record the contents of the *mf6_k* PLIST in file *k.txt*. PLPROC is instructed to record PLIST element values one-to-a-line using 9 significant figures.

```
ptf $
$#p mf6_k.write_in_sequence(format="(1pg16.9)")
```

PLPROC is then instructed to write another file. This file is not required by MODFLOW 6. However it will be used later by the MF62VTK utility to prepare MODFLOW 6 hydraulic property values for importation into PARAVIEW. MF62VTK reads files that contain columns of numbers. However it requires that each of these columns be preceded by a text header. This requires a slightly different template file from that which is required to write a MODFLOW 6 input file. The PLPROC function call is:

```
# -- The following function populates another file with K values. The
#    MF62VTK utility will read this file. MF62VTK requires that columns
#    of numbers possess headers.

write_model_input_file(template_file='k_for_pictures.txt.tpl',      &
                       model_input_file= 'k_for_pictures.txt')
```

The contents of the *k_for_pictures.txt.tpl* PLPROC template file are as follows. Note the "K" header to the column of hydraulic conductivity values.

```
ptf $
K
$#p mf6_k.write_in_sequence(format="(1pg16.9)")
```

We will not produce any pictures just yet because pilot-point-based hydraulic properties are uniform within each layer. Soon we will endow pilot points with random values governed by a variogram. Pictures of the model's hydraulic conductivity field will then be far more interesting.

5.3 Running PLPROC

Run PLPROC using the following command:

```
plproc plproc1.in
```

Functions that are called as the script is run are recorded on the screen. While implementing the scripted commands, PLPROC writes the following files:

- Binary files containing pilot-point-to-model-grid interpolation factors. These are named *factors_layN.dat*, where *N* ranges between 1 and 4;
- *k.txt*, the file from which MODFLOW 6 will read hydraulic conductivities for all model cells;
- *k_for_pictures.txt*, a file that will be used to create a VTK file which PARAVIEW will then use to create coloured pictures of model parameterization.

6. Structural Overlay Features

6.1 Feature Positions

The *pictures* subfolder of the base working folder includes two SURFER BLN files. These are named *initial_aquitard_hole.bln* and *initial_faults.bln*. Complementary files named *initial_aquitard_hole_vertices.txt* and *initial_fault_vertices.txt* provide tables of aquitard hole vertices and fault vertices respectively. These features were digitized using SURFER. However they could have been digitized using any other suitable platform. The above files specify the location of a hole that will be inserted into the layer 3 aquitard, and the locations of two faults that will penetrate layers 2 to 4 of the model grid. If these files are imported into SURFER and plotted over the model grid, the following picture is obtained.

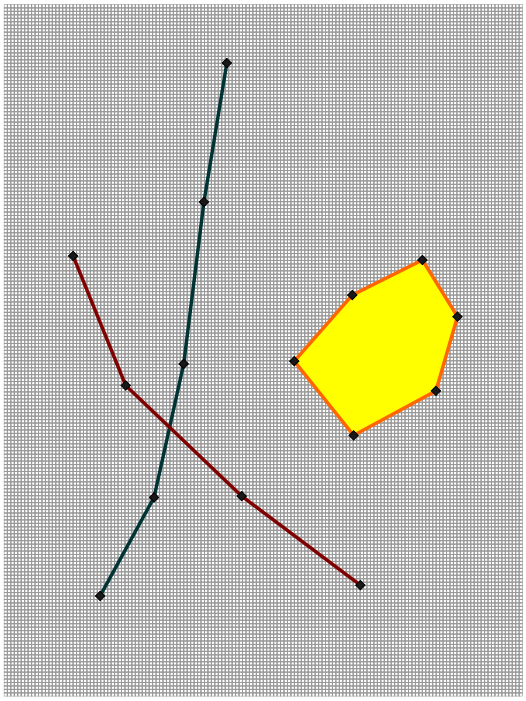


Figure 6.1. A hole and two faults. The hole will be assigned to model layer 3 while the faults will be assigned to model layers 2, 3 and 4. These features were manually digitized. The black triangles depict their vertices.

It is important when digitizing structural overlay features such as these to keep track of the ordering of points which define their vertices. The ordering of vertices which define the aquitard hole on the one hand, and the faults on the other hand, are recorded in files *initial_aquitard_hole_vertices.txt* and *initial_fault_vertices.txt*. Using these files, vertices can be labelled according to their order so that we do not forget which point pertains to which vertex. See Figure 7.1.

6.2 Informing PLPROC of Feature Positions

The base working folder contains two files which record structural overlay feature specifications in a form that PLPROC can read. These files are named *hole_details.dat* and *fault_details.dat*. These were built from the abovementioned BLN files simply by adding some extra columns.

Let's look at *fault_details.dat* first; it is reproduced below. Don't worry about the final column. It is not needed yet. We include it now because we will need it later. The important things to note are the "structure_id" column and the "kh" column. The "structure_id" column lists structure identifiers. These differentiate one fault from the other. Integer values which identify different structures should start at 1 and increase in increments of 1 where new structures are encountered. Counting should start at the top of the file. Structures with higher-valued integer identifiers post-date structures with

lower-valued identifiers; if these structures intersect, the hydraulic properties of younger structures overwrite those of older structures.

Values in the "kh" column of file *fault_details.dat* are hydraulic conductivities. For now, hydraulic conductivity is uniform along each structure, but differs between structures. (Note that the names of column headers in this file are arbitrary; they are there for our benefit, and not for PLPROC's benefit.)

point	easting	northing	structure_id	kh	pos_on_slider
1	3223.4994734995	9153.5041535042	1	0.1	0.5
2	2891.9991419991	7145.0021450021	1	0.1	0.5
3	2599.4988494989	4804.9998049998	1	0.1	0.5
4	2170.4984204984	2874.4978744979	1	0.1	0.5
5	1390.4976404976	1450.9964509965	1	0.1	0.5
6	1000.4972504973	6365.0013650014	2	0.5	0.5
7	1760.998010998	4492.9994929995	2	0.5	0.5
8	3437.9996879997	2893.9978939979	2	0.5	0.5
9	5154.0014040014	1606.9966069966	2	0.5	0.5

Figure 6.2. File *fault_details.dat*.

File *hole_details.dat* is reproduced below. The "structure_id" column is not really needed in this case as there is only one hole. Once again, ignore the "pos_on_slider" column for now. It will be discussed later.

point	easting	northing	structure_id	kh	pos_on_slider
1	6051.0023010023	6306.5013065013	1	0.05	0.5
2	5037.0012870013	5799.5007995008	1	0.05	0.5
3	4198.5004485004	4843.9998439998	1	0.05	0.5
4	5056.5013065013	3771.4987714988	1	0.05	0.5
5	6246.0024960025	4414.9994149994	1	0.05	0.5
6	6558.0028080028	5487.5004875005	1	0.05	0.5

Figure 6.3. File *hole_details.dat*.

Later in this tutorial we will generate random values of hydraulic conductivity for fault and aquifer hole vertices. To do this, templates of the above files are needed. These are named *fault_details.tpl* and *hole_details.tpl*. These have been provided. If you inspect these files you will notice that parameter names have been given to structural feature vertex hydraulic conductivities. They have also been given to the "pos_on_slider" attribute of each feature vertex. This is discussed below.

6.3 Updating the PLPROC Script

PLPROC will now be instructed to read these structure specifications and properties, and to superimpose these properties over those that were introduced to the model through pilot points. See file *plproc2.in*. Additions to file *plproc1.in* are now described.

A new CLIST named *cl_hole* is created by reading file *hole_details.dat*. Complementary PLISTs named *hole_k* and *hole_slider_pos* are populated from the contents of the 5th and 6th columns of this file. An SLIST named *hole_id* is populated from the contents of the 4th column of this file. (Recall from PLPROC documentation that PLISTs contain real numbers whereas SLISTs contain integers. The latter are normally used for selection of subsets of PLISTs.) Ignore the *hole_slider_pos* PLIST for now.

```
# -- Read the file that contains aquifer hole coordinates and K's.

cl_hole = read_list_file(file='hole_details.dat',         &
                         skiplines=1,                     &
                         dimensions=2,                    &
                         id_type=indexed,                 &
                         slist='hole_id';column=4,        &
                         plist='hole_k';column=5,         &
                         plist=hole_slider_pos;column=6)
```

Fault details are read using a similar function call.

```
# -- Read the file that contains fault coordinates and K's.

cl_faults = read_list_file(file='fault_details.dat',        &
                           skiplines=1,                     &
                           dimensions=2,                    &
                           id_type=indexed,                 &
                           slist='fault_id';column=4,       &
                           plist='fault_k';column=5,        &
                           plist=fault_slider_pos;column=6)
```

As is explained in PLPROC documentation, the hydraulic properties of structural overlay features are simultaneously superimposed and blended with hydraulic properties that already populate a model grid. Interpolation factors which govern this blending operation are calculated by function *calc_structural_overlay_factors()*. The blending width is governed by the *conwidth* and *a* arguments of this function. In the present case we make these small. *Conwidth* is 50 m for the faults and 10 m for the aquitard hole. (A *conwidth* value of 50 m guarantees continuity of hydraulic conductivity along a fault despite the fact that it is not parallel to a model grid row or column.) Meanwhile the *a* argument is set to 50 m for both of these features. The larger is *a*, the more "blurred" is the superimposition of structural hydraulic properties on those of the underlying grid. This increases the sensitivity of model-calculated system states at off-feature locations to the properties and positions of these structural features. This can enhance estimability of their positions during history-matching.

As well as superimposing the hydraulic properties of structural features on those with which the model grid is already populated, function *calc_structural_overlay_factors()* also undertakes spatial interpolation of hydraulic properties from feature vertices to points along and within each structural feature. For polylinear features, interpolation is linear between vertices. For polygonal features, spatial interpolation is inverse power of distance (the user nominates the power).

In our present example, structure blending and interpolation factors are calculated using two function calls. These are reproduced below. Note how the aquitard hole is confined to model layer 3, while the two faults are superimposed on layers 2 to 4 of the model grid.

```
# -- Calculate interpolation factors from the structures to the model grid.

calc_structural_overlay_factors(source_clist=cl_hole,                  &
                                structure_type=polygonal,              &
                                target_clist=cl_mf6;select=(layer==3), &
                                file=factors_hole.dat;form='binary',   &
                                inverse_power=2.0,                     &
                                conwidth=10.0,                         &
                                a=50.0)

calc_structural_overlay_factors(source_clist=cl_faults,                &
                                strucid_slist=fault_id,                &
                                structure_type=piecewise_linear,       &
                                target_clist=cl_mf6;select=(layer>1),  &
                                file=factors_faults.dat;form='binary', &
                                conwidth=50.0,                         &
                                a=50.0)
```

Once interpolation/blending factors have been calculated, they can be applied using PLPROC function *interpolate_and_blend_using_file()*. The positioning of this function in the overall PLPROC script (recorded in file *plproc2.in*) is important. Blending of structural feature properties with existing model properties must happen after pilot point parameterisation of the model domain has taken place as this operation overwrites previous parameterization in areas where blending factors are non-zero. The function calls are as follows.

```
# -- Now do the interpolation.

mf6_k=fault_k.interpolate_and_blend_using_file(                          &
                              file=factors_faults.dat;format=binary,     &
                              transform=log,                             &
                              lower_than_target=no)

mf6_k=hole_k.interpolate_and_blend_using_file(                           &
                            file=factors_hole.dat;format=binary,         &
                            transform=log,                               &
                            lower_than_target='no')
```

Introduction of structural features to the model requires no other alterations to the PLPROC script. Hydraulic conductivity values (which include those associated with structural features) are written to files *k.txt* and *k_for_pictures.txt* in the manner described above.

Run PLPROC using the command:

```
plproc plproc2.in
```

6.4 Making Pictures

We will now run MF62VTK so that we can view model hydraulic conductivities in PARAVIEW.

```
Program MF62VTK writes a "legacy" VTK file based on a MODFLOW6 binary grid
  output file and, optionally, associated node data.

Enter name of MODFLOW6 binary grid file: model.dis.grb
- file model.dis.grb read ok.

Enter name for VTK output file: .\pictures\k_field.vtk

Record scalar data in VTK file?  [y/n]: y
Obtain scalar data from tabular file or layer files? [t/l]: t
Enter name of tabular file from which to read cell data: k_for_pictures.txt

Enter column number of data to read (<Enter> if no more): 1
Is this integer or real data? [i/r]: r

Enter column number of data to read (<Enter> if no more): <Enter>

- file k_for_pictures.txt read ok.
- file .\pictures\k_field.vtk written ok.
```

Figure 6.4 shows some of the model's hydraulic conductivity field. See how properties of the second fault overwrite those of the first. See also how blending of structural feature parameter values with model background parameterisation makes the boundaries of the structural features blurry rather than sharp. As stated above, this increases estimability of those parameters which govern the positions and shapes of these features. These parameters will be discussed shortly. Note also that the extent of blurriness can easily be increased by raising the value of the *a* argument of PLPROC function *calc_structural_overlay_factors()*.

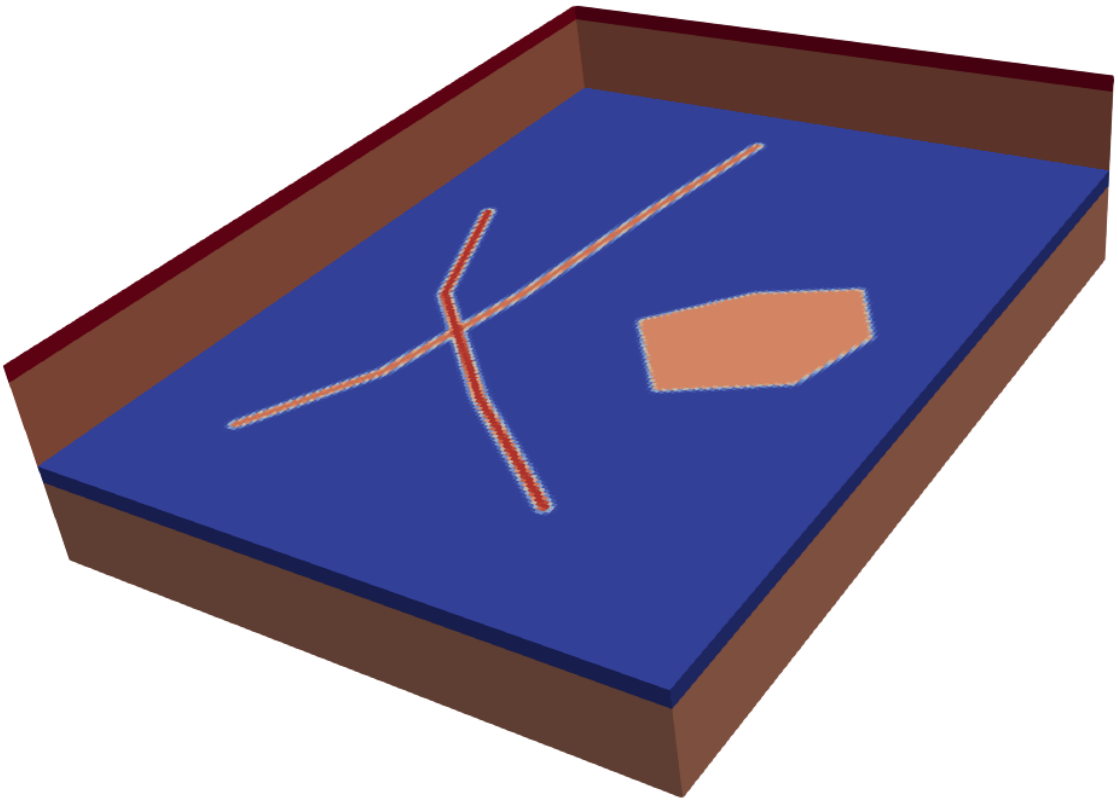


Figure 6.4. Hydraulic conductivity field of the model; warmer colours indicate higher K. Layers 1 and 2 have been removed.

Figure 6.5 shows a view from underneath the model. The longer fault is less distinct than the shorter fault in this picture because its hydraulic conductivity is nearly the same as the background hydraulic conductivity of the lowermost model layer.

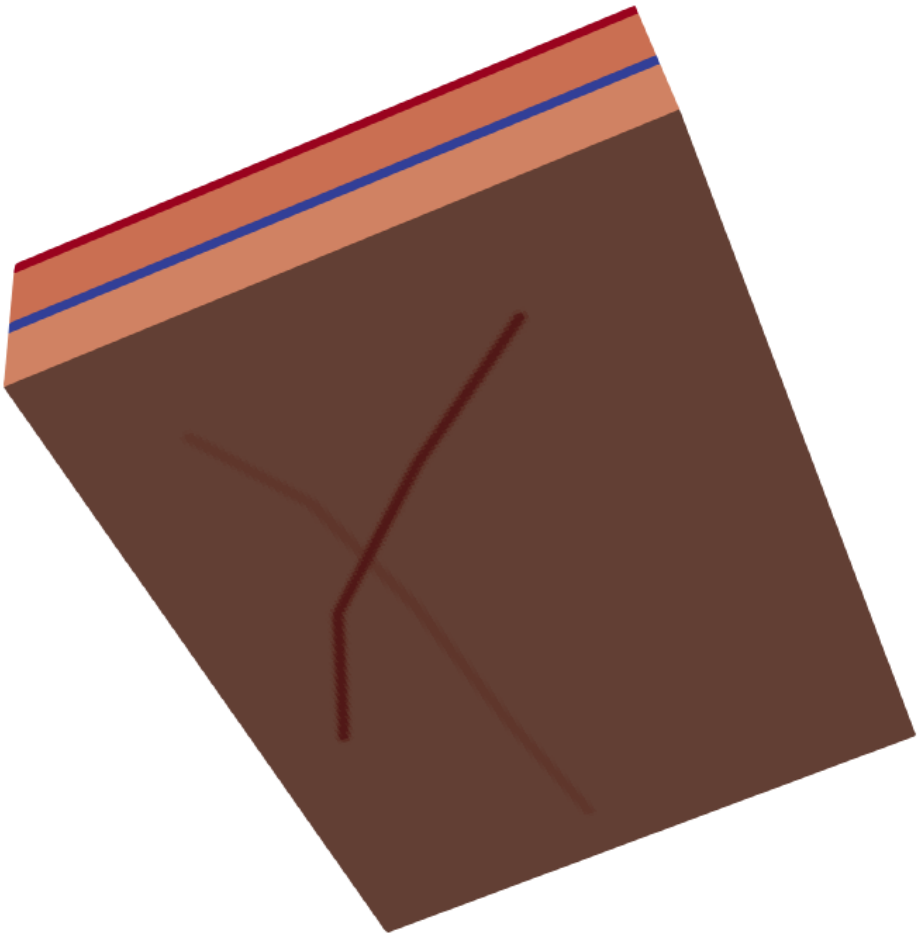


Figure 6.5. View from underneath the model.

At this stage our pictures look rather boring. This is because pilot points that are assigned to each model layer have the same value of hydraulic conductivity throughout the layer, and because hydraulic conductivities assigned to all vertices of each structural feature are the same for each feature.

Figure 6.6 shows the top of model layer 4. Obviously the hole in the aquitard is confined to the aquitard.

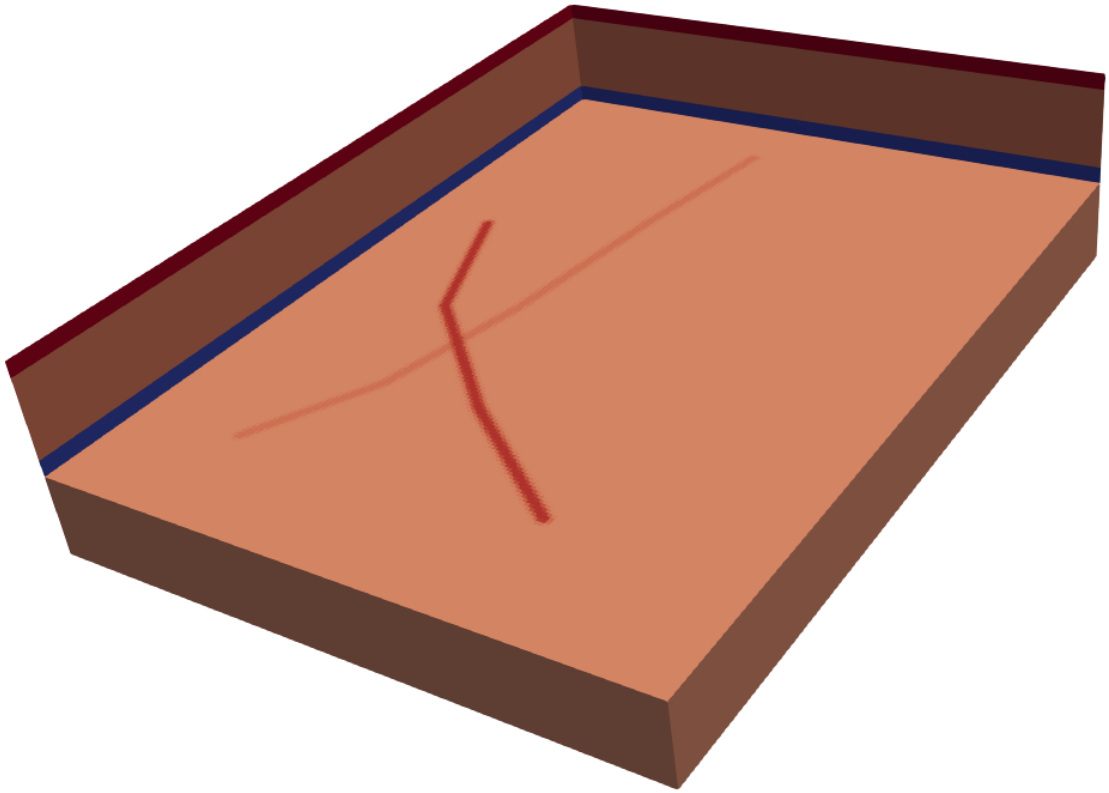


Figure 6.6. Hydraulic conductivity field in model layer 4.

7. Moving Structural Overlay Features

7.1 Sliders

All vertices of all structural features that have been introduced to the model domain will now be assigned sliders. A slider is a polylinear feature along which the vertex of a structural feature can glide. The position of a structural feature vertex on its slider is denoted by a real number that ranges from 0.0 to 1.0. Ordering of points that define a slider is important; a distance of 0.0 corresponds to the first slider-defining point while a distance of 1.0 corresponds to the last point which defines a particular slider.

Like any other set of points, the vertices of a slider are assigned to a CLIST. They can be digitized using software such as SURFER or a GIS. Normally only a few points are required to define each slider; two is the minimum. When digitizing sliders, it is wise to ensure that each slider passes through (or close to) the structural feature vertex that must slide along it. However, it does not matter if the digitized slider does not pass right through the vertex; PLPROC will move the structural feature's vertex to the slider when vertex-to-slider assignment is performed.

For the present tutorial, sliders were digitized using SURFER. BLN files named *hole_sliders.bln* and *fault_sliders.bln* are provided in the *pictures* subfolder. Sliders appear as thin black lines in Figure 7.1. Meanwhile, the vertices of structural features are labelled according to their order in Figure 7.1. This is the same order as their order of appearance in files that define them, namely *hole_details.dat* and *fault_details.dat*.

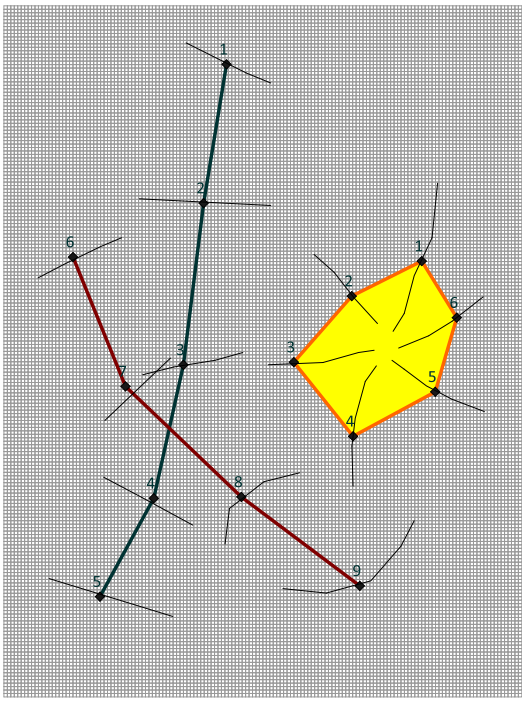


Figure 7.1. Structural features, along with the sliders along which their vertices can move. The vertices of structural features are numbered according to their order of appearance in the files that define them.

7.2 Informing PLPROC of Sliders

Files *hole_sliders.dat* and *fault_sliders.dat* are provided in the working folder. These provide the same slider vertex coordinates as the abovementioned BLN files. However these files are suitable for reading by PLPROC. PLPROC assigns slider vertices to two CLISTS, one of which pertains to the aquitard hole and one of which pertains to the two faults.

The first part of file *hole_sliders.dat* is shown below.

```
point_id     x                    y                slider_track_id
1            6280.167819424       7433.4735078537    1
2            6199.5892264487      6643.8032966959    1
3            5941.7377289278      6095.8688644639    1
4            5796.6962615723      5547.934432232     1
5            5635.5390756217      5290.082934711     1
6            4491.3230553726      6402.06751777      2
7            4781.4059900837      6144.2160202491    2
8            5071.4889247947      5773.5544925628    2
9            5409.9190152909      5402.8929648764    2
10           3830.5785929753      4806.6113768593    3
etc
```

Figure 7.1. The first part of file *hole_sliders.dat*.

The fourth column of file *hole_sliders.dat* associates each slider vertex with the vertex of a structural feature, in this case the aquitard hole. Note that slider vertices must be provided in the correct order in a file such as *hole_sliders.dat*. Firstly, the order of slider vertices along which the vertex of any structural feature must slide defines the sliding direction. Secondly, all slider vertices assigned to the first structural feature vertex must appear before slider vertices that are assigned to the second structural feature vertex, and so on.

The protocol for file *fault_sliders.dat* follows that of file *hole_sliders.dat*; the latter file is not illustrated herein.

7.3 Updating the PLPROC Script

See file *plproc3.in*. The following two calls to function *read_list_file()* follow the function calls which create the *cl_hole* and *cl_faults* CLISTs. These functions create CLISTs which represent the sliders. These CLISTs are named *cl_h_sliders* (for the aquitard hole) and *cl_f_sliders* (for the faults). An SLIST is associated with each of these CLISTs. These are named *hole_sl_track_id* and *fault_sl_track_id* respectively. These SLISTs host the integers which link each slider to a structural overlay feature vertex.

```
# -- Read the files that contain slider CLIST specifications.

cl_h_sliders = read_list_file(file='hole_sliders.dat',          &
                              skiplines=1,                      &
                              dimensions=2,                     &
                              id_type=indexed,                  &
                              slist='hole_sl_track_id';column=4)

cl_f_sliders = read_list_file(file='fault_sliders.dat',         &
                              skiplines=1,                      &
                              dimensions=2,                     &
                              id_type=indexed,                  &
                              slist='fault_sl_track_id';column=4)
```

(Note the use of commas and semicolons in the above, and other, PLPROC functions. Commas separate function arguments whereas semicolons separate function subarguments.)

Next, the vertices of the structural features need to be placed on their sliders. This is done using function *slide_clist_vertices_on_clist()*. When a structural feature vertex is placed on a slider, its position on the slider must be provided. Recall that this number must be in the range 0.0 to 1.0; one such number must be provided for each structural feature vertex. These numbers are provided in the *hole_slider_pos* and *fault_slider_pos* PLISTs. Recall that these PLISTs were read when the *cl_hole* and *cl_fault* CLISTs were created. They comprise column 6 of files *hole_details.dat* and *fault_details.dat*. In accordance with the contents of these files, all of these slider positions are presently 0.5. Shortly they will be assigned random variables. (Remember that we have PEST template files that correspond to files *hole_details.dat* and *fault_details.dat*.)

```
# -- Position the hole and fault vertices on sliders.

slide_clist_vertices_on_clist(                                          &
                        vertex_sliding_clist=cl_hole;                   &
                        position_on_slider_plist=hole_slider_pos,       &
                        slider_track_clist=cl_h_sliders;                &
                        slider_track_id_slist=hole_sl_track_id,         &
                        blnfile=.\pictures\hole.bln;                    &
                        sliding_structure_id_slist=hole_id;             &
                        sliding_structure_type=polygonal)

slide_clist_vertices_on_clist(                                          &
                        vertex_sliding_clist=cl_faults;                 &
                        position_on_slider_plist=fault_slider_pos,      &
                        slider_track_clist=cl_f_sliders;                &
                        slider_track_id_slist=fault_sl_track_id,        &
                        blnfile=.\pictures\faults.bln;                  &
                        sliding_structure_id_slist=fault_id;            &
                        sliding_structure_type=piecewise_linear)
```

As is recorded in PLPROC documentation, and as is illustrated by the above function calls, function *slide_clist_vertices_on_clist()* optionally writes a SURFER BLN file in which the new positions of slider-mounted structural features are recorded. These are their positions after their vertices have been placed on slider tracks at their user-specified locations along these tracks (in this case 0.5 for all structural feature vertices). Figure 7.2 shows the new positions of the structural features.

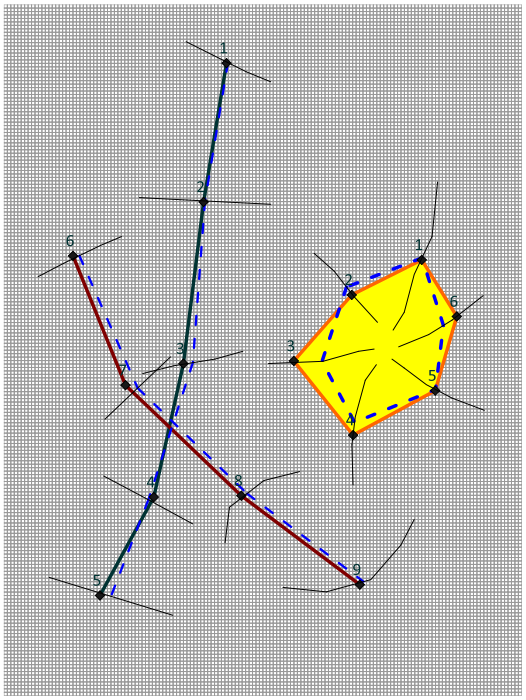


Figure 7.2. The dashed blue lines show structural overlay features mounted on sliders, with a slider position of 0.5 assigned to each vertex of each structure.

Run PLPROC using the command:

```
plproc plproc3.in
```

to make sure that all is working as it should.

8. Sliding Sliders

8.1 Moving the Aquitard Hole

The previous section described how the vertices of structural overlay features can be mounted on sliders. Hence the two faults can change their positions, while the hole in the aquitard can expand, shrink and change its shape in other ways. We will now allow the hole to move as it changes its shape. This is accomplished by moving the group of sliders to which the hole's vertices are attached as a single unit. Recall that aquitard hole vertex sliders comprise the *cl_h_sliders* CLIST. We will now inform PLPROC that it must collectively slide the entirety of this *cl_h_sliders* CLIST along a slider that belongs to a new CLIST that we will name *cl_hg_slider*. ("g" stands for "global".)

Digitizing the new slider produces a BLN file named *\pictures\global_hole_slider.bln*. Its contents are depicted in Figure 8.1 (as the long red slider in the east of the model domain). This new slider was digitized to pass through the end vertex of the *cl_h_sliders* slider along which the first vertex of the aquitard hole slides. This slider end vertex will be affixed to the global slider.

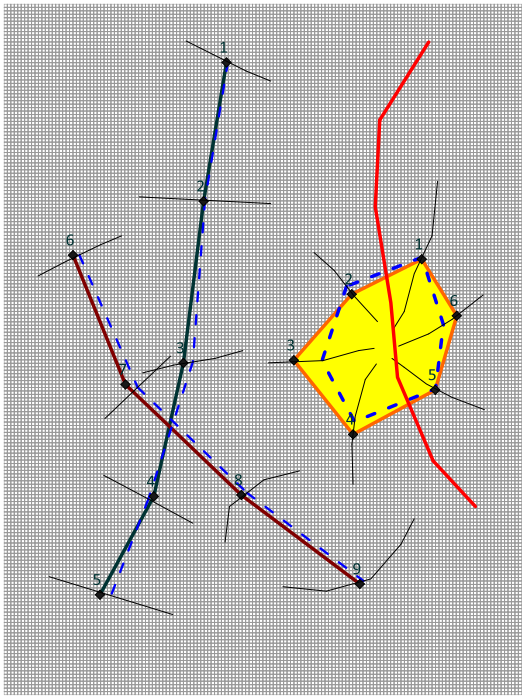


Figure 8.1. Sliders assigned to vertices of the aquifer hole collectively glide along the red slider. The anchor point to the red global slider is the last point of the slider that passes through the first vertex of the aquitard hole.

8.2 Informing PLPROC of Sliding Sliders

Specifications of the global slider are provided in PLPROC-ready form in file *global_hole_slider.dat*. See below. These are read into a CLIST using the PLPROC *read_list_file()* function, the same function that has been used to create most CLISTs that feature in the current PLPROC script; see file *plproc4.in*.

```
point_id    x                   y
1           6151.2420706636     9447.9383322359
2           5442.150452481      8319.8380305818
3           5377.6875781008     7062.8119801674
4           5603.3076384316     5676.8601809924
5           5700.001950002      4597.1070351236
6           6215.7049450438     3388.4281404943
7           6828.102251656      2727.6836780969
```

Figure 8.2. File *global_hole_slider.dat*.

The call to PLPROC function *read_list_file()* follows. Note that this function call must precede the call to function *slide_clist_vertices_on_clist()*, as the positions of hole vertex sliders must be settled before they are used to change the shape of the aquitard hole.

```
# -- Read the details of the global hole slider.

cl_hg_slider = read_list_file(file='global_hole_slider.dat',     &
                              skiplines=1,                       &
                              dimensions=2,                      &
                              id_type=indexed)
```

The *cl_h_sliders* CLIST (along which aquitard hole vertices slide) is next attached to the *cl_hg_slider* CLIST. Vertex number 5 of the former CLIST slides along the latter CLIST. Its position on the global slider is denoted using a real number between 0.0 and 1.0; in this case we use 0.5. The PLPROC function is *slide_entire_clist_on_clist()*.

```
# -- Slide all of the aquitard hole sliders along the global hole slider.

slide_entire_clist_on_clist(                                     &
                            slider_track_clist=cl_hg_slider,     &
                            sliding_clist=cl_h_sliders,          &
                            sliding_clist_vertex_number=5;       &
                            position_on_slider_clist=0.5)
```

Our PLPROC script is nearly complete - but not quite. We want the aquifer hole to slide along the global hole slider rather than maintain a fixed position of 0.5 on this global slider. Hence its proportional position along this slider must be a PEST-accessible parameter. This is most easily accomplished if PLPROC reads this relative position from an external file. A PEST template of this file can then identify the relative position as a parameter.

File *global_hole_pos.dat* is a file from which PLPROC can read a so-called SCALAR. A file such as this can define many SCALARs; the names of SCALARs must appear in the first column of such a file while SCALAR values must appear in the second column. In the present case we read only one SCALAR. The file from which it is read (i.e. file *global_hole_pos.dat*) is shown below. A template of this file named *global_hole_pos.tpl* is also provided.

```
global_hole_pos          0.5
```

Figure 8.3. File *global_hole_pos.dat*.

File *plproc5.in* reads the SCALAR file *global_hole_pos.dat* using function *read_scalar_file()*. The function call is as follows.

```
# -- Read a scalar file to obtain the position of the aquitard hole on its global
#    slider.

read_scalar_file(file=global_hole_pos.dat,                       &
                 namecolumn=1,                                   &
                 valuecolumn=2)
```

The call to function *slide_entire_clist_on_clist()* can then be modified to use this SCALAR instead of the fixed value of 0.5.

```
# -- Slide all of the aquitard hole sliders along the global hole slider.

slide_entire_clist_on_clist(                                     &
                            slider_track_clist=cl_hg_slider,     &
                            sliding_clist=cl_h_sliders,          &
                            sliding_clist_vertex_number=5;       &
                            position_on_slider_clist=global_hole_pos)
```

Run PLPROC using the command:

```
plproc plproc5.in
```

To make sure that everything works as it should.

9. Making it all Stochastic

9.1 Template Files

We have accomplished much. We have defined structural features, and we have endowed them with the ability to move and change shape. So we will now move them, at the same time as we will provide them, and the material which hosts them, with random realisations of hydraulic properties.

Table 9.1 lists template files which we have gathered along the way.

Table 9.1. Template files and the parameters that they host.

Template file	Corresponding model input file	Parameters
pp.tpl	*pp.dat*	Pilot point kh for model layers 1 to 4.
hole_details.tpl	*hole_details.dat*	• kh at each aquifer hole vertex; • position of each aquifer hole vertex along its slider.
fault_details.tpl	*fault_details.dat*	• kh at each fault vertex; • position of each fault vertex along its slider.
global_hole_pos.tpl	*global_hole_pos.dat*	Position of aquifer hole sliders along their global slider.
k.txt.tpl	*k.txt*	Used by PLPROC; not by PEST.
k_for_pictures.txt.tpl	*k_for_pictures.txt*	Used by PLPROC; not by PEST.

In a moment we will provide a PEST control file which features the first four of these template files, and the parameters that they host. This will allow us to generate random values of these parameters. The "model" that will be run through this PEST control file is PLPROC. (We will not run MODFLOW 6.)

9.2 "Observations"

If we are going to build a PEST control file, then we need some observations. Because PLPROC is the model, these observations should be PLPROC outputs. So we will add some lines to the end of our PLPROC script that calculate the geometric average of hydraulic conductivity within each model layer. PEST will read these averages using an appropriate instruction set. (We will not be using these observations to optimize parameters. They are there because PEST requires observations.)

See file *plproc6.in*. The lines that are added to *plproc5.in* to make *plproc6.in* follow. First the logs of all elements of the *mf6_k* PLIST are calculated. The average of these logs within each model layer is then taken; these averages are assigned to four SCALARs named *avk1*, *avk2*, *avk3* and *avk4*. The values of these SCALARs are then recorded in a file named *plproc_calculated_stats.dat* using a PLPROC template file named *plproc_calculated_stats.tpl*.

```
# -- Now we calculate some "observations" for PEST to read.
#    These are the geometric average of K in each layer.

log_mf6_k = log10(mf6_k)
avk1=log_mf6_k.stat(statistic=mean,select=(layer==1))
avk2=log_mf6_k.stat(statistic=mean,select=(layer==2))
avk3=log_mf6_k.stat(statistic=mean,select=(layer==3))
avk4=log_mf6_k.stat(statistic=mean,select=(layer==4))
avk1=10^avk1
avk2=10^avk2
avk3=10^avk3
avk4=10^avk4

# -- Record the statistics that have just been calculated in a file.

write_model_input_file(template_file=plproc_calculated_stats.tpl,      &
                       model_input_file=plproc_calculated_stats.dat)
```

9.3 A PEST Control File

See file *case.pst*. Some features of this PEST control file are as follows.

- *case.pst* features all parameters that are named in the first four template files that are listed in Table 9.1.
- All of these parameters are given appropriate initial values and lower/upper bounds.
- The "model" featured in the "model command line" section of *case.pst* is PLPROC; PLPROC runs the script that is recorded in file *plproc6.in*.
- Geometrically averaged hydraulic conductivities in each model layer are the four observations that are listed in the "observation data" section of *case.pst*. They are given dummy "observed values" in this file.
- Model-calculated observations are read from the PLPROC output file *plproc_calculated_stats.dat* using an instruction file named *plproc_calculated_stats.ins*.
- The NOPTMAX control variable (at the beginning of the 9th line of *case.pst*) is set to 0. Hence PEST is instructed to run the model once and then cease execution.

Recall that the main purpose of the PLPROC script is to write a MODFLOW 6 input file named *k.txt* which lists hydraulic conductivities in all model cells. Recall also that PLPROC writes a similar file named *k_for_pictures.txt*. This file is used by the MF62VTK utility to write a VTK file that can be imported into PARAVIEW so that hydraulic conductivities can be viewed in three dimensions.

Check the entire PEST input dataset using PESTCHEK.

```
pestchek case
```

Ignore its warning messages. Now run "the model" once using PEST.

```
pest case
```

9.4 Characterising Parameter Stochasticity

Shortly we will generate random realisations of all parameters. However before we can do this, we must characterise parameter stochasticity.

We will assume that the mean values of all parameters are their initial values as provided in file *case.pst*.

We will use a variogram to describe spatial correlation of pilot point parameters within each model layer. See file *struct.dat* for specifications of this variogram. It is exponential (VARTYPE=2) with a sill of 0.0625 in log space. This is equivalent to a standard deviation of 0.25 in log space (a quarter of an

order of magnitude). The “a” value of the exponential variogram (equal to about a third of its range) is 2000 m.

Before we can generate random parameter realisations of pilot point parameters, we must build a covariance matrix for these parameter types. We will use a single covariance matrix to describe spatial parameter correlation in all model layers as the disposition of pilot points is the same in each layer. The PPCOV utility supplied with the PEST Groundwater Utility Suite will be used to build this covariance matrix. However before we can use this utility, we must alter our pilot points file (*pp.dat*) slightly so that PPCOV can read it. See file *pp_for_ppcov.dat*. The format of the latter file is in accordance with older standards set by the PEST Groundwater Utilities. The file contains five columns of data. The first column endows each pilot point with a name, while the second and third columns contain pilot point eastings and northings. Pilot point zone numbers and values occupy the fourth and fifth columns. In the present case, all pilot points in each layer belong to the same zone. The last column of a pilot points file (i.e. the column that contains pilot point values) is ignored when building a covariance matrix.

Run PPCOV, responding to its prompts as follows:

```
Program PP2COV prepares a covariance matrix file for pilot point parameters
  based on a geostatistical structure file.

Enter name of pilot points file: pp_for_ppcov.dat
 - data for 336 pilot points read from pilot points file pp_for_ppcov.dat

Enter minimum allowable separation for points in same zone: 0

Enter name of structure file: struct.dat

Enter structure to use for pilot point zone 1: struc1

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

Filling covariance matrix....
- file cov.mat written ok.
```

A file name *param.unc* has been provided in the working folder. This is a “parameter uncertainty file”. Programs of the PEST and PEST++ suites rely on a file of this type to specify prior parameter uncertainties. As is apparent from an inspection of this file, the uncertainties of some parameters can be characterised using individual parameter standard deviations, while the uncertainties of other groups of parameters can be characterised using covariance matrices. It is important to note that if a parameter is log-transformed in a PEST control file, then the uncertainty that is awarded to this parameter in a parameter uncertainty file must pertain to its log (to base 10).

Full specifications of a parameter uncertainty file are provided in Part 2 of the PEST manual.

File *param.unc* uses the “first parameter, last parameter” protocol to link elements of a covariance matrix to individual parameters. These descriptors refer to the locations of parameters in the “parameter data” section of a PEST control file. It is assumed that the order of parameter occurrence in a PEST control file is the same as their order of occurrence in the covariance matrix file. The latter is inherited from the order of pilot point occurrence in the pilot point file that was read by PPCOV when it wrote the covariance matrix file.

10. Generating Random Sets of Parameters

10.1 The RANDPAR Utility

We now have a PEST control file and a complementary parameter uncertainty file. The RANDPAR utility can be used to generate random realisations of PEST parameters. RANDPAR stores the random parameter sets which it generates in individual parameter value files. Run RANDPAR by typing its name at the screen prompt. Respond to RANDPAR's prompts as follows.

```
RANDPAR Version 17.5. Watermark Numerical Computing.

Enter name of existing PEST control file: case.pst
- 1375 parameters read from file case.pst.
- 1375 of these are adjustable.

Use (log)normal or (log)uniform distrib for param generation? [n/u]: n
Compute means as existing param values or range midpoints? [e/m]: e
Respect parameter ranges? [y/n]: y

Enter name of parameter uncertainty file: param.unc
- covariance matrix file cov.mat read ok.
- covariance matrix file cov.mat read ok.
- covariance matrix file cov.mat read ok.
- covariance matrix file cov.mat read ok.
- parameter uncertainty file param.unc read ok.

Enter name of parameter ordering file (<Enter> if none): <Enter>

Enter filename base for parameter value files: random
How many of these files do you wish to generate? 5

Enter integer random number seed (<Enter> if default): <Enter>
- file random1.par written ok.
- file random2.par written ok.
- file random3.par written ok.
- file random4.par written ok.
- file random5.par written ok.
```

10.2 Random MODFLOW 6 Parameter Fields

Our next task is to use these random parameter realisations to construct random hydraulic conductivity fields for the MODFLOW 6 model. Recall that our MODFLOW 6 model reads hydraulic conductivity values from a file named *k.txt*. PLPROC also writes a file named *k_for_pictures.txt* wherein model hydraulic conductivity values are more accessible to PARAVIEW for display. We will run PEST (which runs the PLPROC "model") to populate these files. But first, initial parameter values in the PEST control file *case.pst* must be replaced with random parameter values. This can be done using the PARREP utility that is supplied with PEST.

Run PARREP using the command:

```
parrep random1.par case.pst case1.pst
```

Now run PEST using *case1.pst* as the PEST control file:

```
pest case1
```

When "the model" has finished execution, run MF62VTK in the same way that we did previously:

```
Program MF62VTK writes a "legacy" VTK file based on a MODFLOW6 binary grid
  output file and, optionally, associated node data.

Enter name of MODFLOW6 binary grid file: model.dis.grb
- file model.dis.grb read ok.

Enter name for VTK output file: .\pictures\k_field.vtk
```

```
Record scalar data in VTK file?  [y/n]: y
Obtain scalar data from tabular file or layer files? [t/l]: t
Enter name of tabular file from which to read cell data: k_for_pictures.txt

Enter column number of data to read (<Enter> if no more): 1
Is this integer or real data? [i/r]: r

Enter column number of data to read (<Enter> if no more): <Enter>

- file k_for_pictures.txt read ok.
- file .\pictures\k_field.vtk written ok.
```

To make the re-running of MF62VTK a little easier, the above keyboard responses can be placed into a keyboard response file. A file of this type named *mf62vtk.in* has been prepared for you. MF62VTK can then be run using the command:

```
mf62vtk < mf62vtk.in
```

The "legacy" VTK file *k_field.vtk* that was written by MF62VTK can now be imported into PARAVIEW in order to visualize hydraulic conductivities (and thereby the locations of structural overlay features) in three dimensions. See the following figures. If you look carefully, you can detect spatial variability of background kh fields incurred by pilot point stochasticity. However hydraulic conductivity variability induced by the presence of structural overlay features is much greater than this.

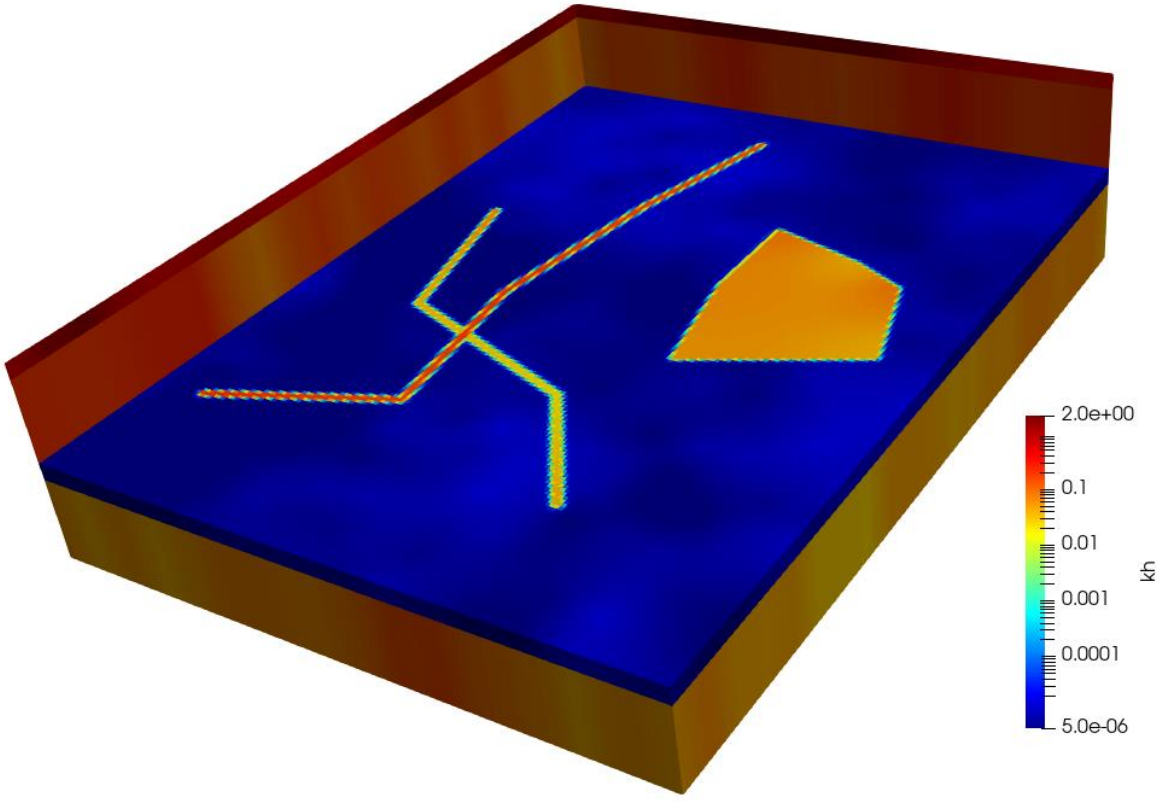


Figure 10.1a. Kh field generated using *random1.par*.

The above process can be repeated using files *random[2-5].par* to obtain four more kh fields. Note the variability in the locations of the two faults and the aquifer hole between these hydraulic conductivity realisations. Note the spatial variability of hydraulic conductivity along the lengths of the faults, and within the aquifer hole.

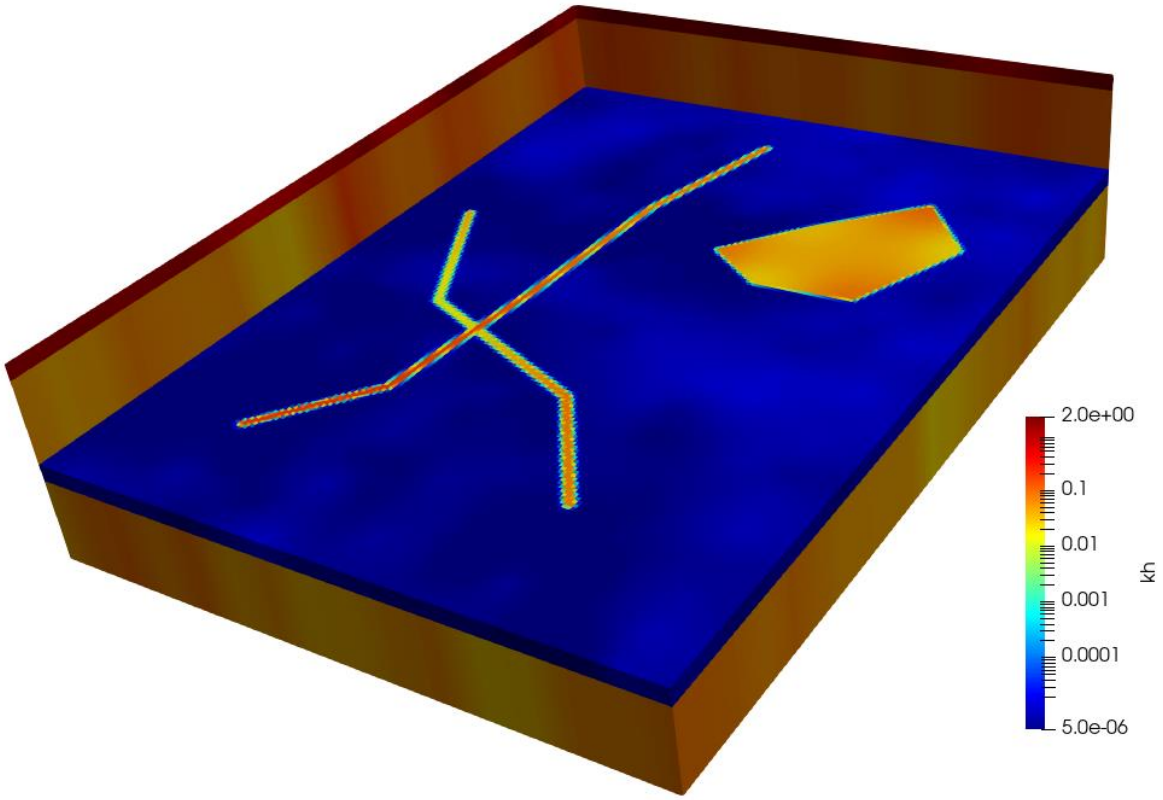


Figure 10.1b. Kh field generated using *random2.par*.

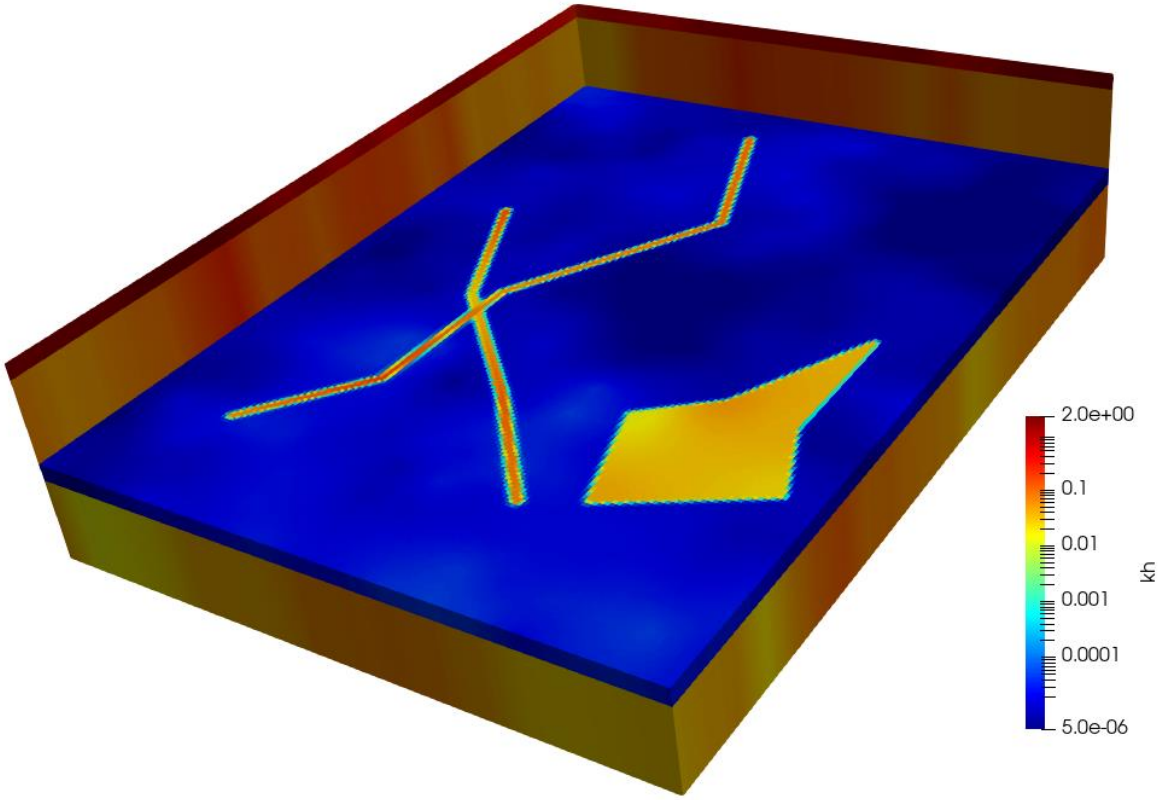


Figure 10.1c. Kh field generated using *random3.par*.

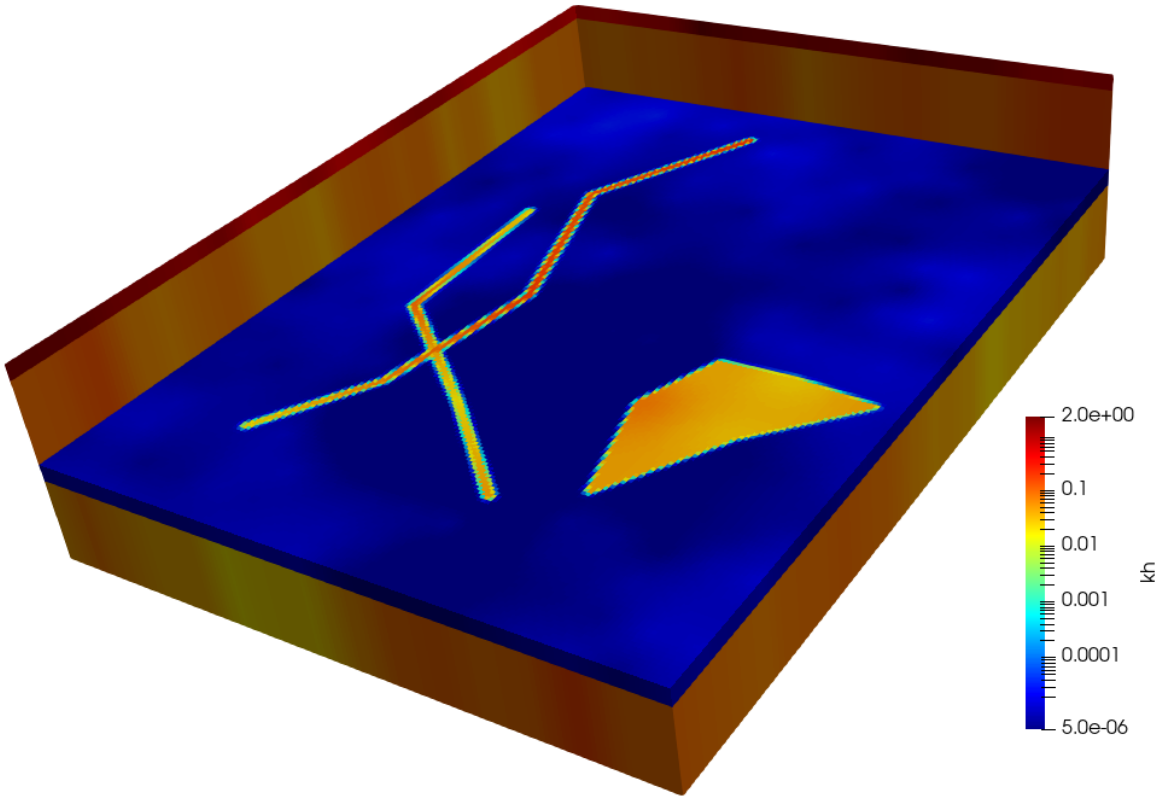


Figure 10.1d. Kh field generated using *random4.par*.

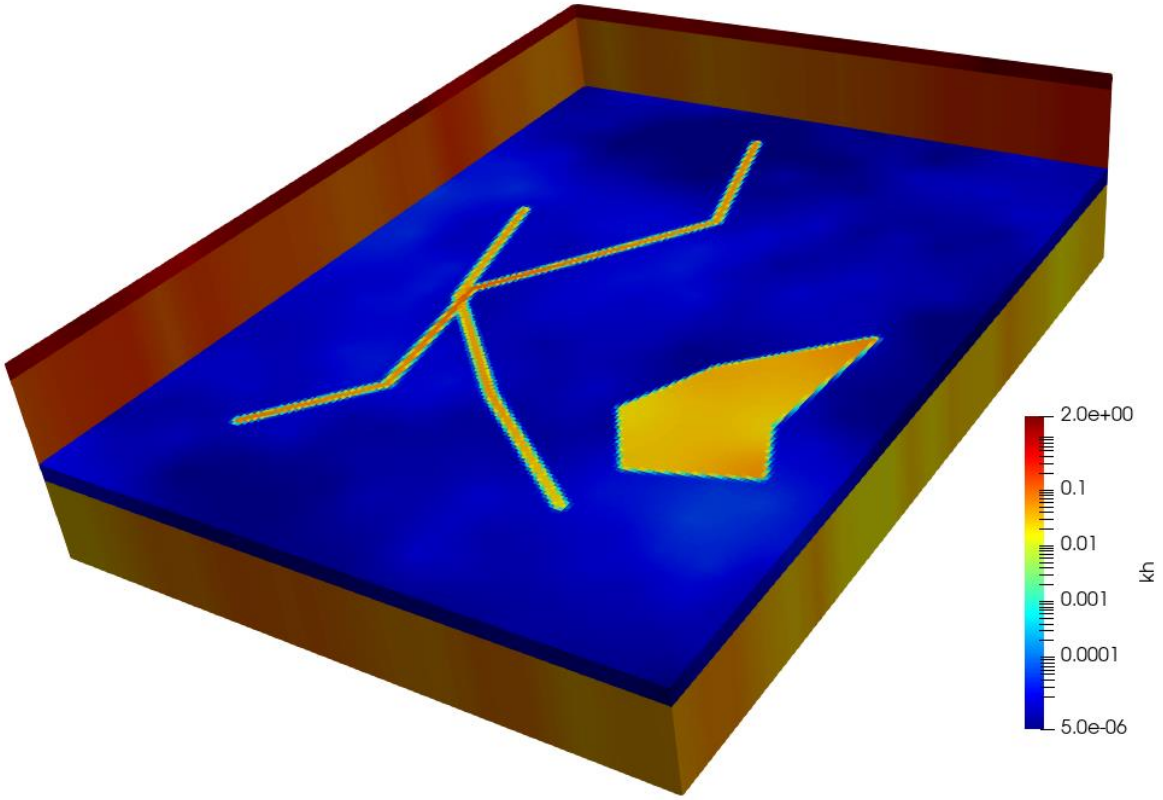


Figure 10.1e. Kh field generated using *random5.par*.

10.3 Taming the Aquifer Hole

Inspection of Figures 10.1 reveals that the aquifer hole can sometimes adopt weird shapes. This is because the locations of its vertices along their respective sliders are independently random. Hence one vertex can slide in one particular direction along its slider, while its neighbour may slide in the opposite direction along its slider. Ungeological star shapes may therefore emerge.

This situation can be rectified if spatial correlation is introduced to the subset of parameters which define aquifer hole vertex locations along their respective sliders. This is achieved if these parameters are provided with a covariance matrix that expresses spatial correlation between them. To make life easy, we will use the same descriptor of spatial correlation as that which was used for pilot point parameters. This is recorded in the structure file *struct.dat*. We will use PPCOV to build a covariance matrix file based on the variogram that is specified in this file for these vertex-along-slider position parameters. But before using PPCOV we need to build a file of aquitard hole vertex coordinates that PPCOV can read. See file *hole_details_for_ppcov.dat*. This is slightly modified from *hole_details.dat*; the header line has been removed.

Run PPCOV by typing its name at the command-line prompt, and then responding to its prompts as follows:

```
Program PP2COV prepares a covariance matrix file for pilot point parameters
  based on a geostatistical structure file.

Enter name of pilot points file: hole_details_for_ppcov.dat
 - data for 6 pilot points read from pilot points file
   hole_details_for_ppcov.dat

Enter minimum allowable separation for points in same zone: 0

Enter name of structure file: struct.dat

Enter structure to use for pilot point zone 1: struc1

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

Filling covariance matrix....
- file cov_hole_sliderpos.mat written ok.
```

The new covariance matrix file *cov_hole_sliderpos.mat* must take its place in a revised parameter uncertainty file. See file *param1.unc*; this is slightly modified from *param.unc* – see the COVARIANCE_MATRIX block at its top and the absence of *pos*_h* parameters from the ensuing STANDARD_DEVIATION section of this file.

Now use RANDPAR to generate another set of random parameter fields. We will name the new set of RANDPAR-generated parameter value files *hrandpar*.par*.

```
RANDPAR Version 17.5. Watermark Numerical Computing.

Enter name of existing PEST control file: case.pst
- 1375 parameters read from file case.pst.
- 1375 of these are adjustable.

Use (log)normal or (log)uniform distrib for param generation? [n/u]: n
Compute means as existing param values or range midpoints? [e/m]: e
Respect parameter ranges? [y/n]: y

Enter name of parameter uncertainty file: param1.unc
- covariance matrix file cov_hole_sliderpos.mat read ok.
- covariance matrix file cov.mat read ok.
- covariance matrix file cov.mat read ok.
- covariance matrix file cov.mat read ok.
- covariance matrix file cov.mat read ok.
- parameter uncertainty file param1.unc read ok.
```

```
Enter name of parameter ordering file (<Enter> if none): <Enter>

Enter filename base for parameter value files: hrandom
How many of these files do you wish to generate? 5

Enter integer random number seed (<Enter> if default): <Enter>
- file hrandom1.par written ok.
- file hrandom2.par written ok.
- file hrandom3.par written ok.
- file hrandom4.par written ok.
- file hrandom5.par written ok.
```

A new set of pictures of MODFLOW 6 parameter fields can be obtained by repeating the following sequence of commands and importing *k_field.vtk* into PARAVIEW on each occasion.

```
parrep hrandomN.par case.pst case1.pst
```

where *N* is 1 to 5.

```
pest case1
```

```
mf62vtk < mf62vtk.in
```

The outcomes of this process are depicted in the following pictures.

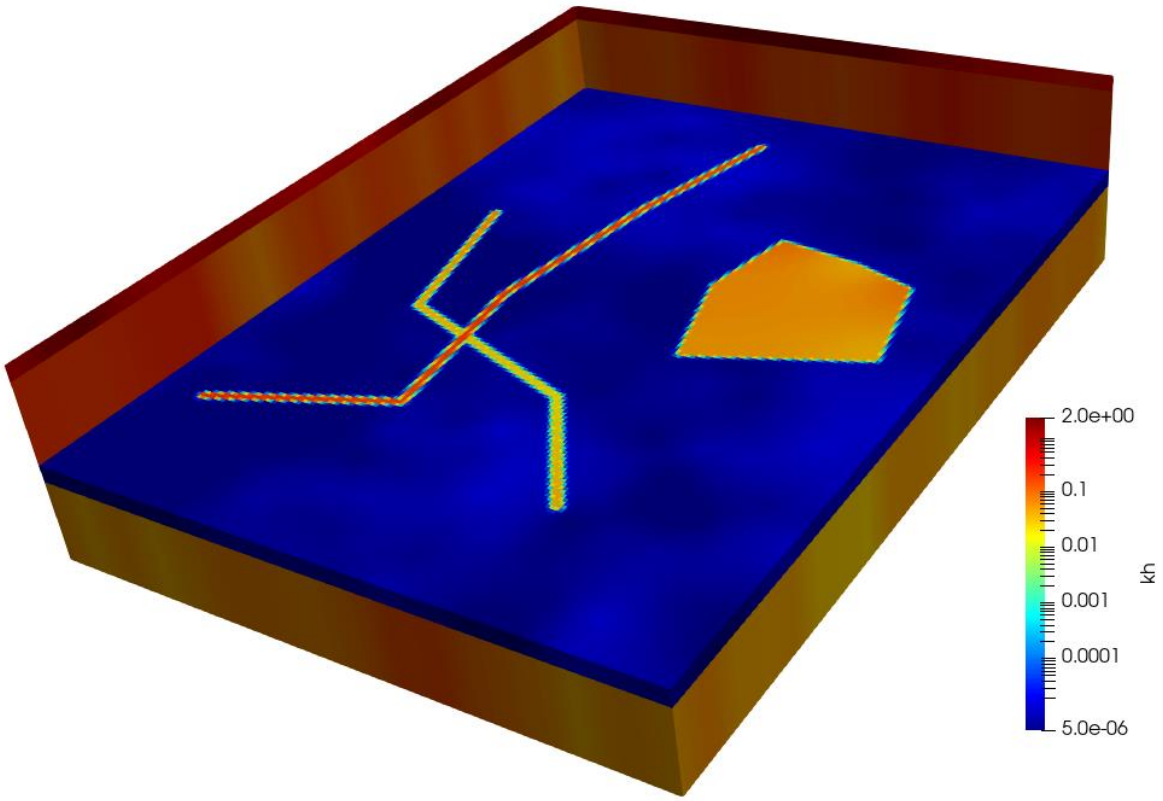


Figure 10.2a. Kh field generated using *hrandom1.par*.

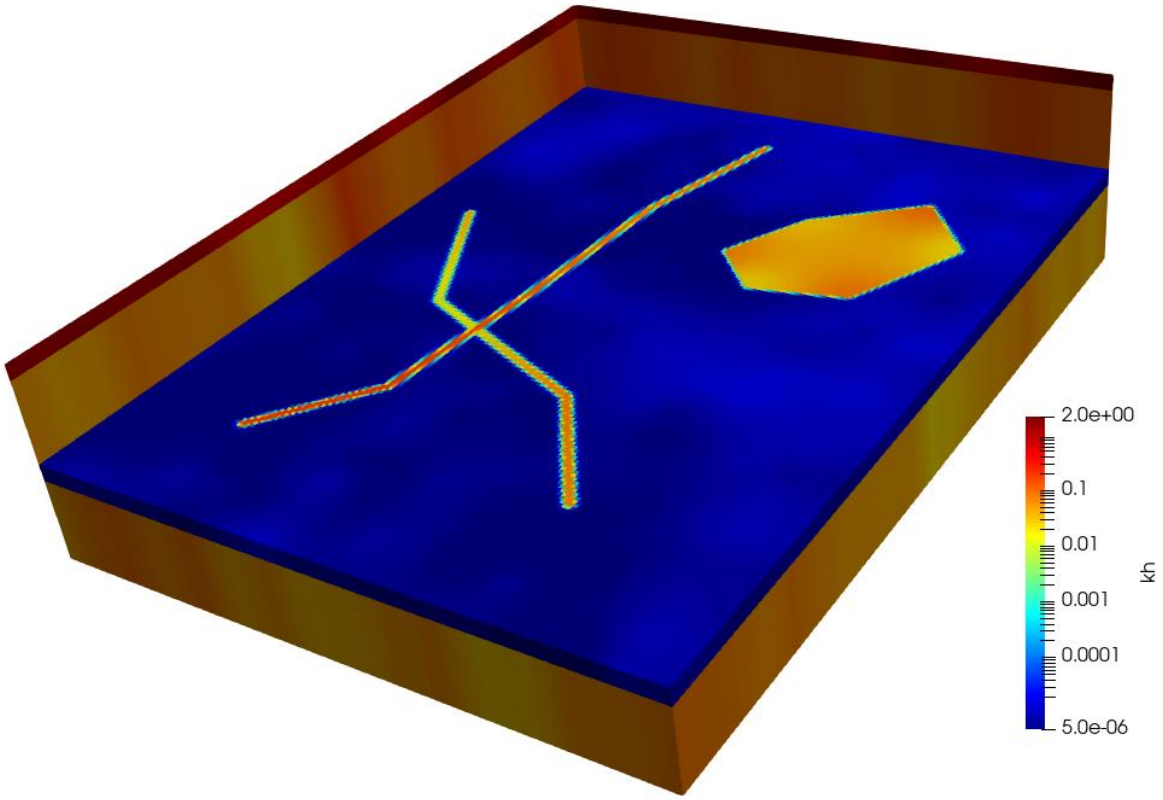


Figure 10.2b. Kh field generated using *hrandom2.par*.

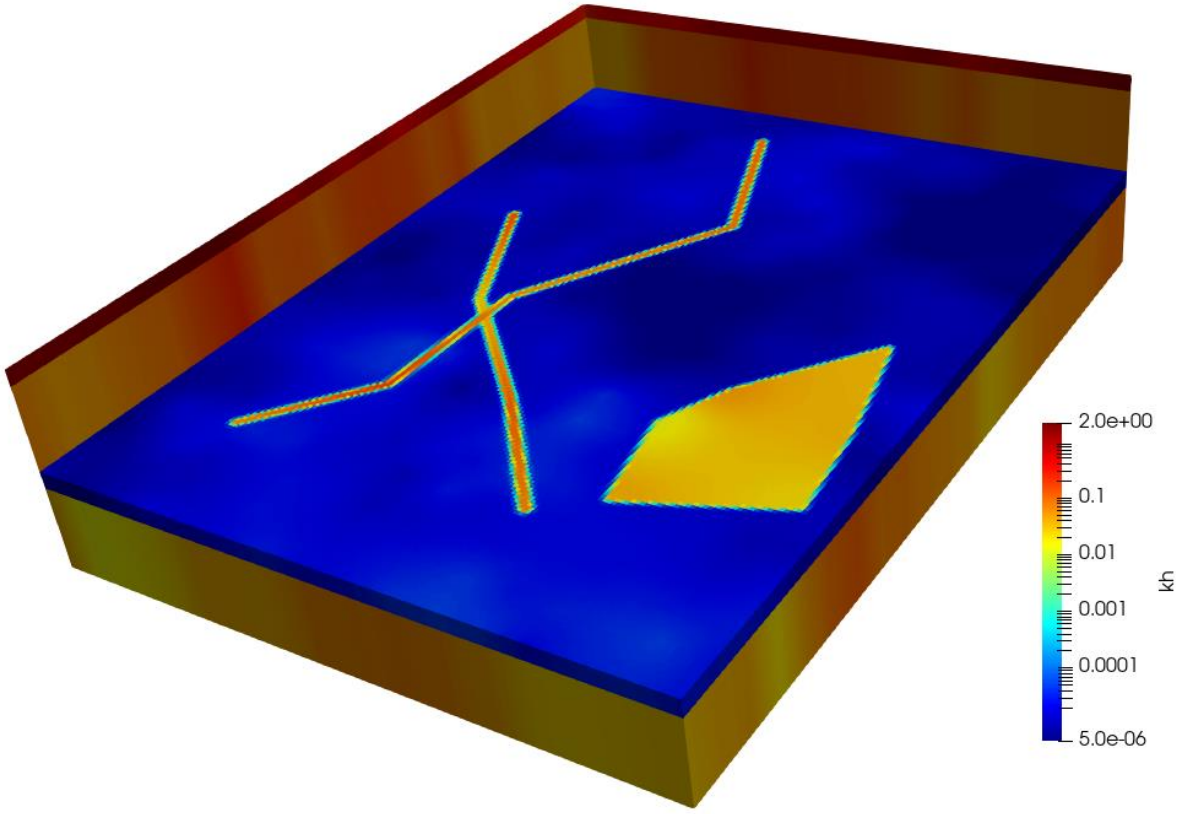


Figure 10.2c. Kh field generated using *hrandom3.par*.

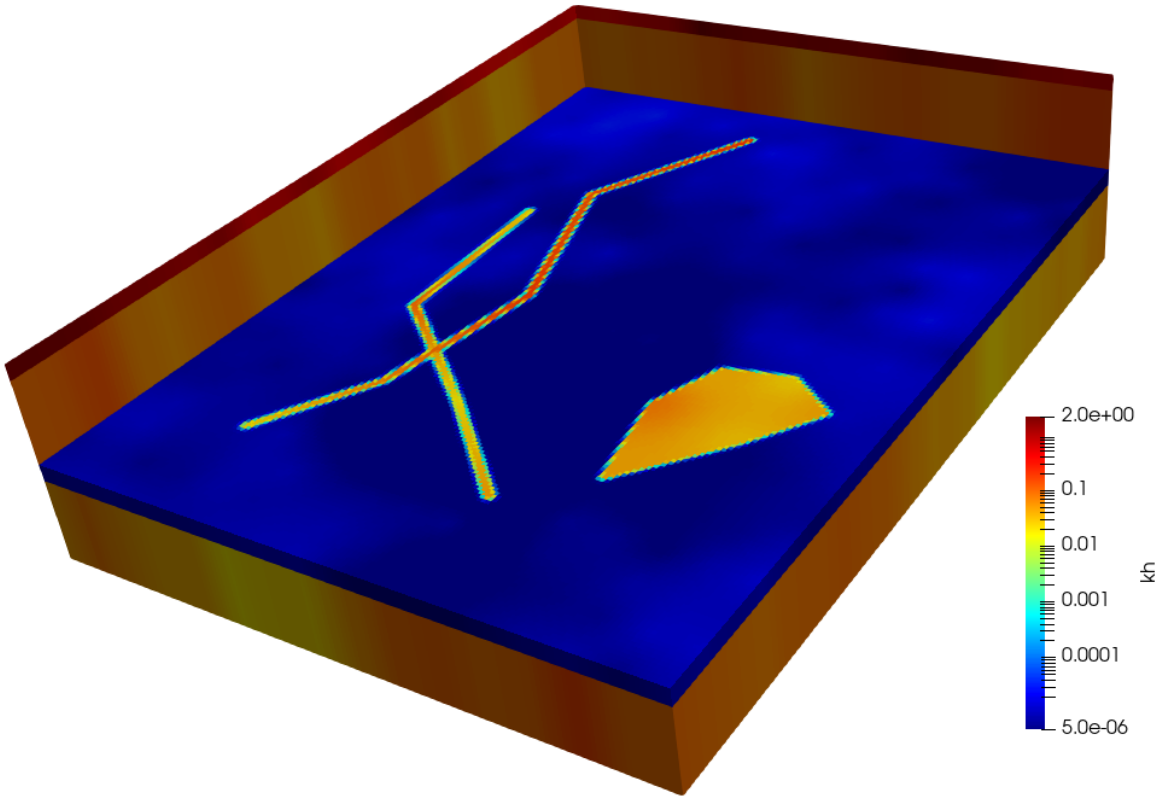


Figure 10.2d. Kh field generated using *hrandom4.par*.

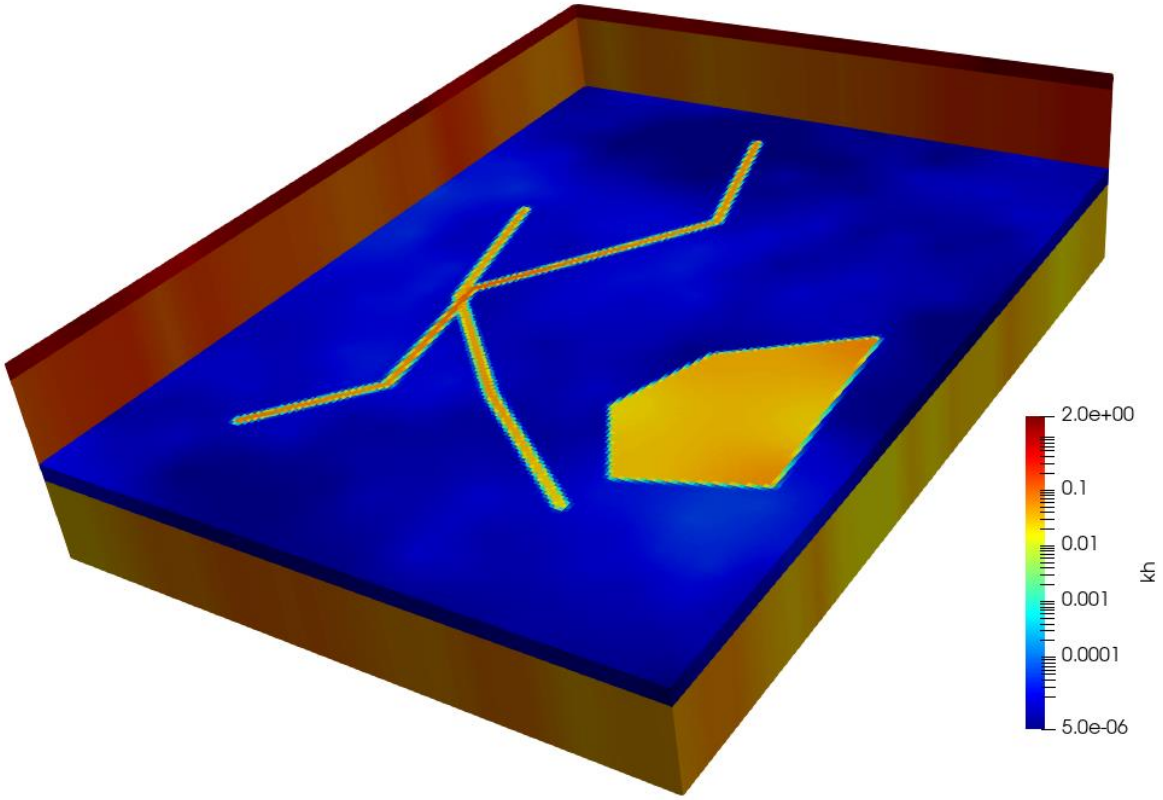


Figure 10.2e. Kh field generated using *hrandom5.par*.