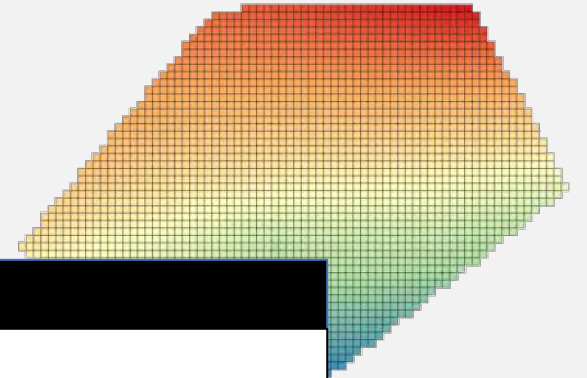
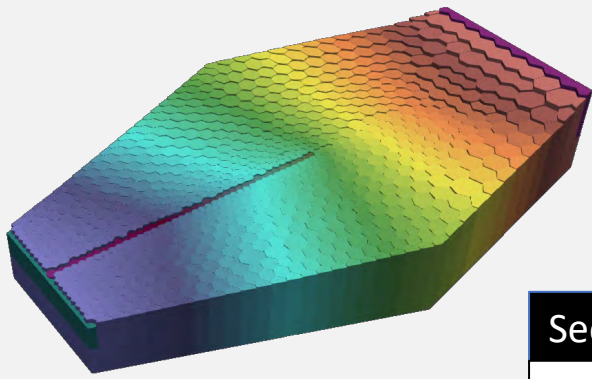


Using AI to help with model construction, history-matching, data assimilation and uncertainty analysis

by
John Doherty

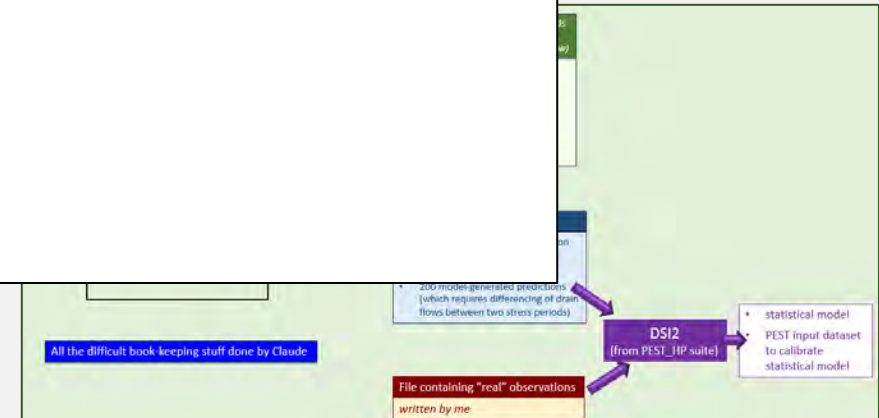
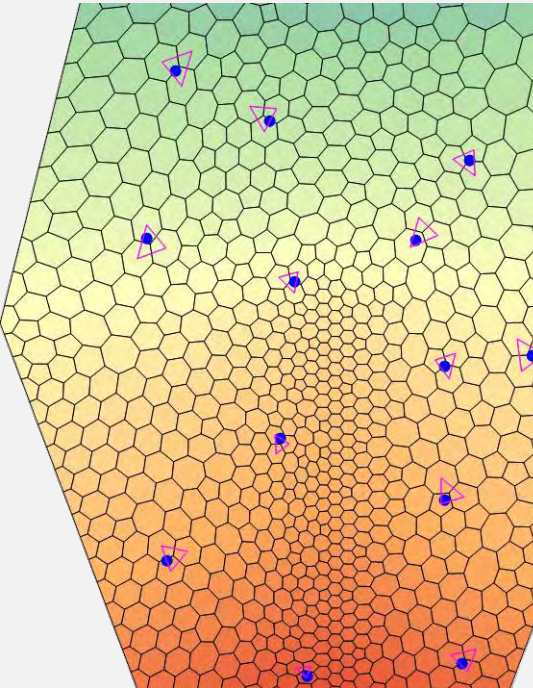
This slideshow documents some of my experiences in getting acquainted with Claude.
Examples starts off simple and get more complicated.

Play this slideshow rather than viewing it so that you can see the animations.



Sections in this slideshow cover the following

- Construction of a simple groundwater model from scratch
- Reading contents of binary MODFLOW 6 output files and exporting them to GIS and 3D visualization packages
- Generating stochastic multiplier fields and inspecting stochastic head fields
- Automatic slideshow generation
- Working with a Voronoi grid
- Writing stand-alone utility programs
- Generating nonstationary stochastic fields
- Automated running of a model many times and collecting results
- Data space inversion
- Installing a network of pilot points
- Hierarchical parameterization
- Preparing for a PEST run



Ideas for a Simple Model for Fast Spring Capture Zone Analysis

What I gave Claude

- A file containing the coordinates of the vertices of a polygon
- The head at each vertex

120 m



110 m



105 m



80 m



What I told Claude

- Create a uniform DIS grid inside the polygon
- Maximum of 50 cells in column/row directions
- Active inside the polygon
- Inactive outside the polygon
- CHD boundary along the line joining each vertex
- Head varies linearly along the boundary

```
name,x,y,elevation
```

```
point1,46478.405965315, 770521.48698359, 120
```

```
point2,45132.718282024, 768903.25495938, 110
```

```
point3,47091.630732385, 767932.31574486, 90
```

```
point4,48846.134927056, 769312.07147076, 105
```

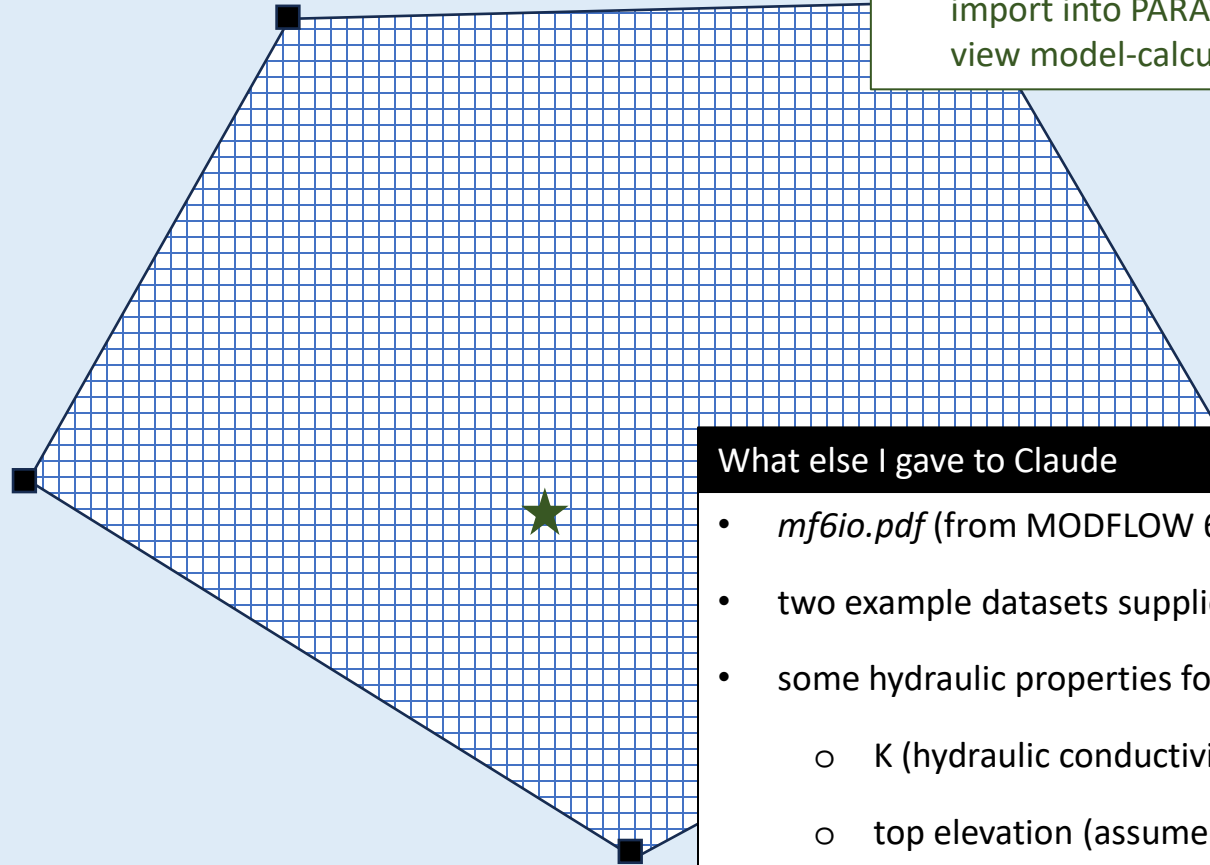
```
point5,48198.842117372, 770555.55502621, 125
```

What I gave Claude

- A file containing the coordinates of a spring

What I told Claude

- Emplace a DRN boundary condition at this location
- Make the conductance high so that water can escape easily
- Now build a complete MODFLOW 6 GWF input dataset
- Also, once the model has run, write a VTK file (that I can import into PARAVIEW) that allows me to see the grid and view model-calculated heads



```
name,x,y,elevation
spring,46887,768869,100.0
```

What else I gave to Claude

- *mf6io.pdf* (from MODFLOW 6 documentation)
- two example datasets supplied with downloaded MODFLOW 6
- some hydraulic properties for the single layer model
 - K (hydraulic conductivity)
 - top elevation (assumed uniform)
 - bottom elevation (assumed uniform)
- a report of MODFLOW 6 errors on the two occasions that they occurred

Files that Claude wrote

24/06/2026	04:35 PM	247	mfsim.nam
24/06/2026	04:35 PM	3,759	mymodel.chd
24/06/2026	04:35 PM	18,800	mymodel.dis
24/06/2026	04:35 PM	145	mymodel.drn
24/06/2026	09:40 AM	45,460	mymodel.hds
24/06/2026	04:35 PM	89	mymodel.ic
24/06/2026	04:35 PM	216	mymodel.ims
24/06/2026	09:40 AM	10,609	mymodel.lst
24/06/2026	04:35 PM	252	mymodel.nam
24/06/2026	04:35 PM	163	mymodel.npf
24/06/2026	04:35 PM	168	mymodel.oc
24/06/2026	04:35 PM	118	mymodel.rcha
24/06/2026	04:35 PM	146	mymodel.tdis
24/06/2026	09:04 AM	624,732	mymodel_heads.vtk

Other files that Claude wrote

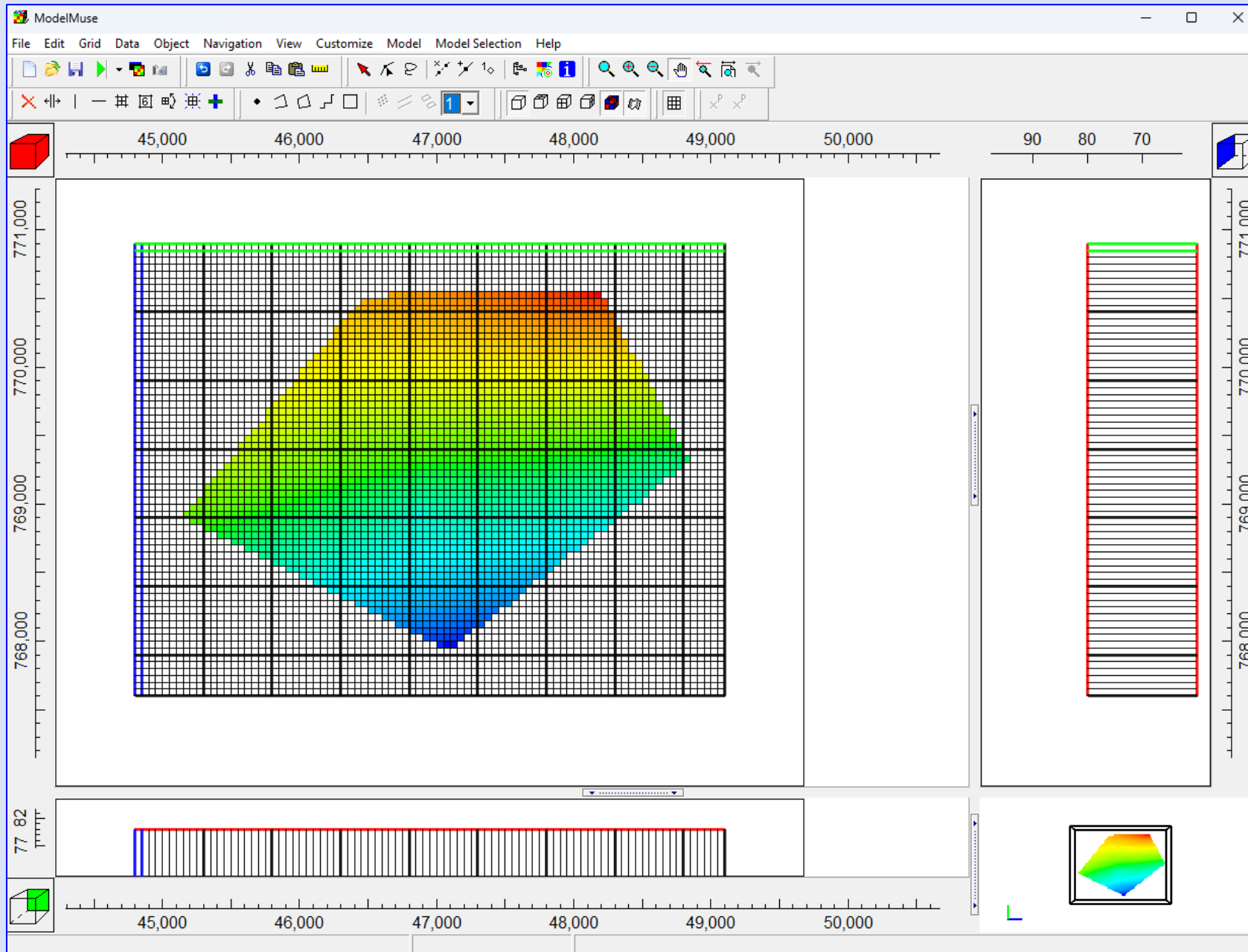
24/06/2026	10:59 AM	4,119	CLAUDE.md
24/06/2026	11:02 AM	9,126	SKILL.md
24/06/2026	10:59 AM	4,010	modflow6-skill.skill

Allows Claude to remember how I like doing things

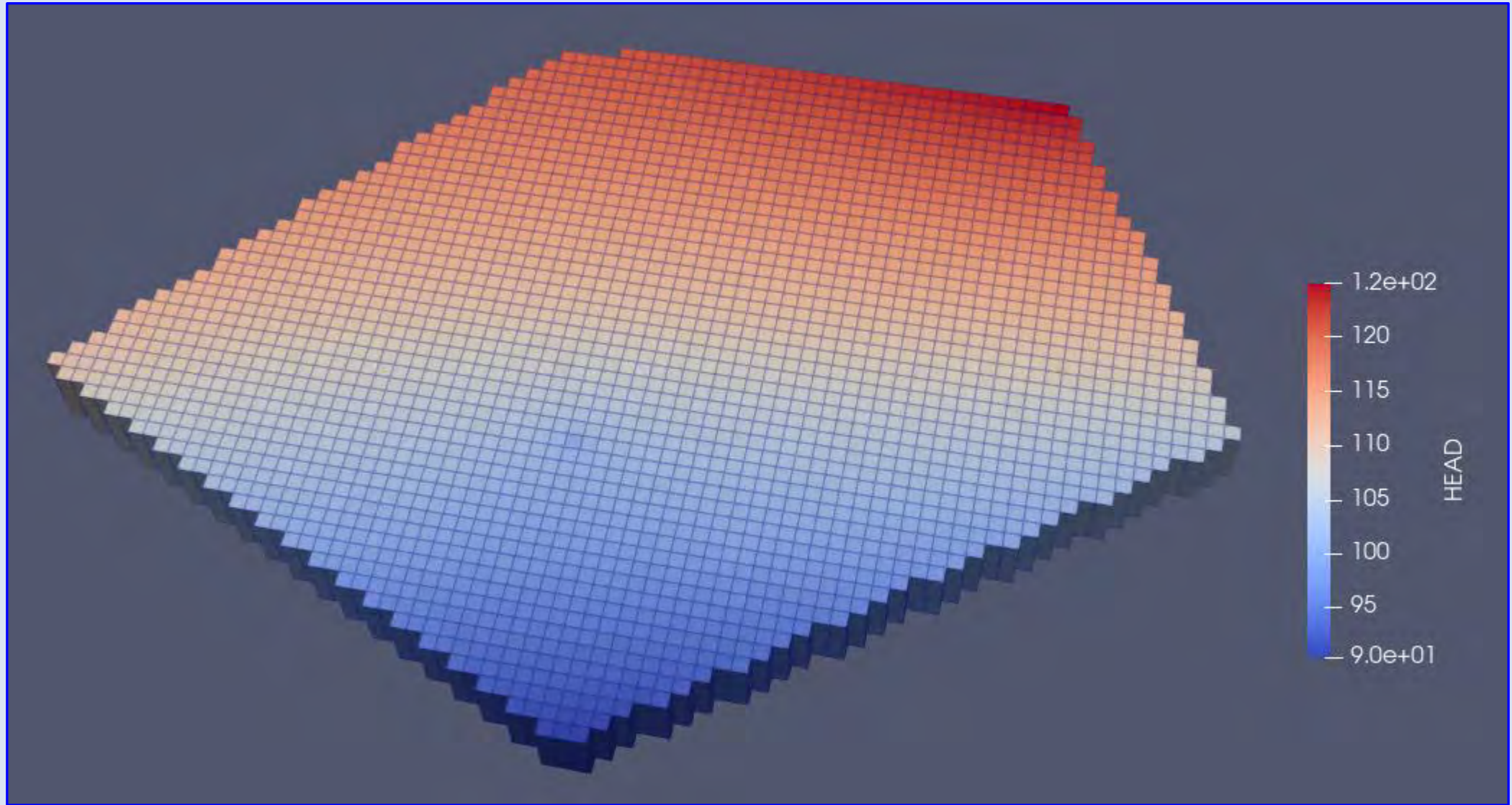
Allows Claude to do this whole thing more efficiently next time

(Binary) Can be given to other people to allow them to do similar things efficiently themselves

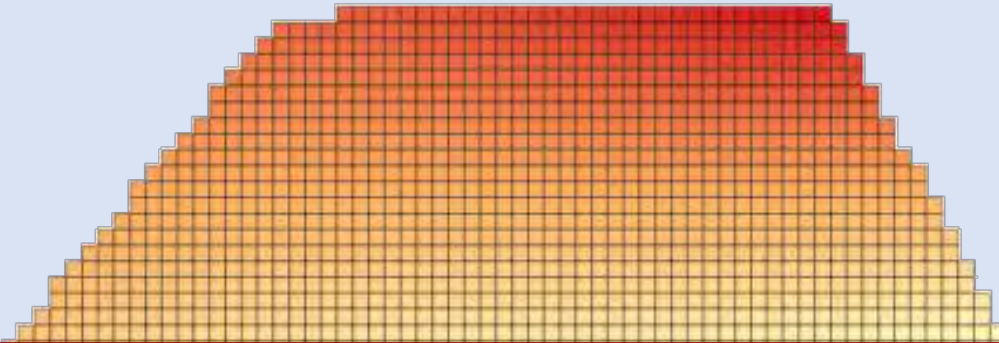
Model and model-computed heads imported into ModelMuse



Model heads imported into PARAVIEW
using the VTK file that Claude wrote



Model heads displayed in a GIS
using MF62GIS from PEST Groundwater Utility Suite



How Claude built the model

- Wrote a Python script. (BTW, there are no calls to FloPy in this script.)
- Ran and tested the script on a “sandbox” on my machine.
- Gave me the script (after I asked for it) so that I could modify it if I wanted to.
- Asked me if I would like the script written in FORTRAN (because it knows that I prefer a compiled rather than interpreted language).

Using the MODFLOW 6 PRT package

Then I gave Claude

- Two examples of use of the PRT (i.e. particle) package that comes with the downloaded version of MF6

I asked Claude to:

- assign a zone number of 1 to all cells in the model domain except the DRN cell that represents the spring
- assign a zone number of 2 to the cell containing the DRN
- release a particle in every cell of the model domain
- write a complete input dataset for the PRT package

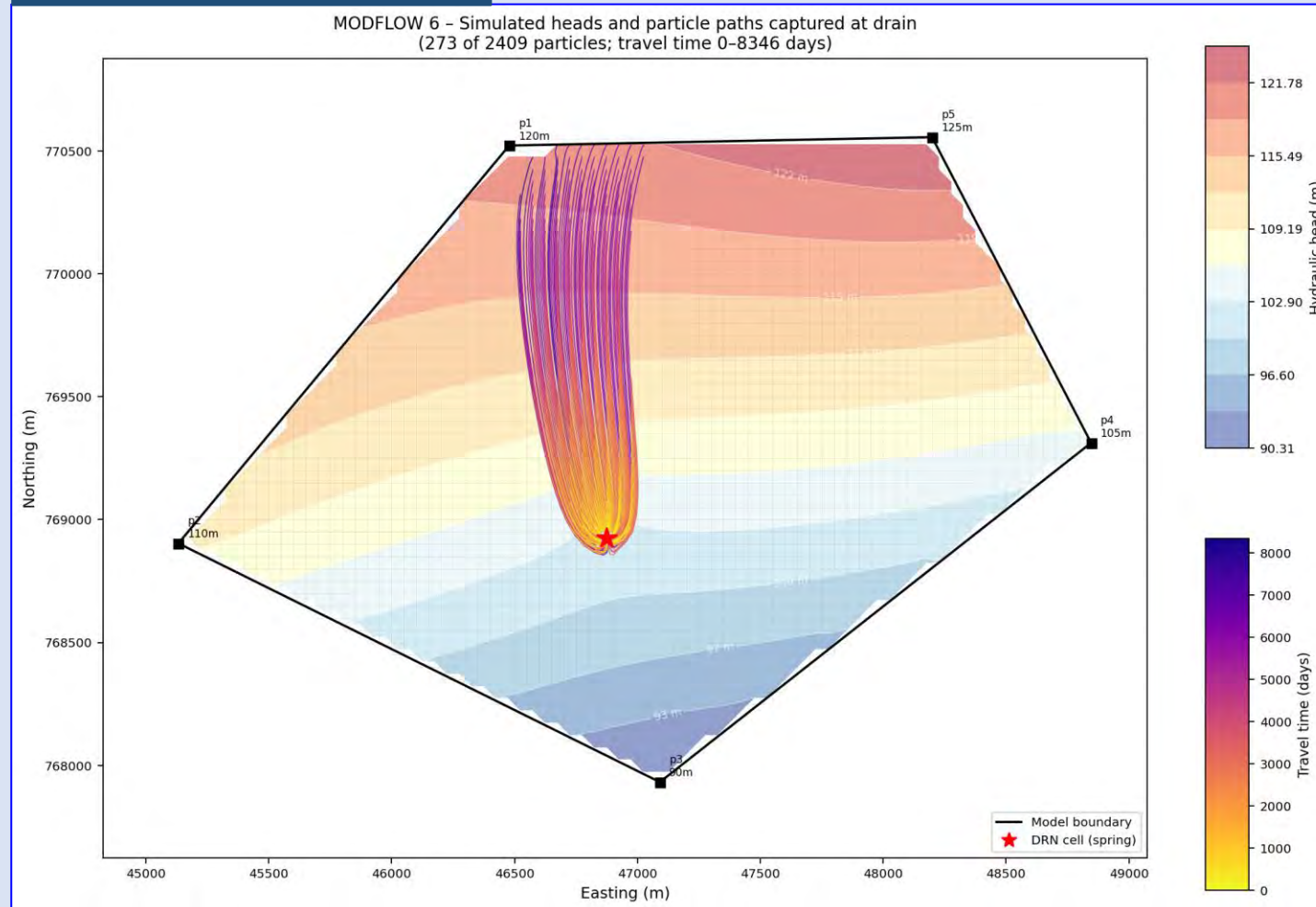
Files that Claude wrote

24/06/2026	10:57 AM	8,341	gen_prt.py
24/06/2026	09:43 AM	7,010	mfsim.lst
24/06/2026	09:36 AM	250	mfsim.nam
24/06/2026	09:36 AM	17,639	mymodel-prt.dis
24/06/2026	09:36 AM	0	mymodel-prt.ems
24/06/2026	09:36 AM	133	mymodel-prt.fmi
24/06/2026	09:43 AM	9,988	mymodel-prt.lst
24/06/2026	09:36 AM	17,420	mymodel-prt.mip
24/06/2026	09:36 AM	199	mymodel-prt.nam
24/06/2026	09:36 AM	1,932	mymodel-prt.oc
24/06/2026	09:42 AM	102,169	mymodel-prt.prp
24/06/2026	09:36 AM	135	mymodel-prt.tdis
24/06/2026	09:36 AM	5,038	prt_to_vtk.py

After running MODFLOW 6, I asked Claude

- Provide a CSV file where only the travel paths/times of those particles that terminate in the DRN cell are represented
- Using data in this file, provide a plot in which particle paths are coloured by residence time

Picture produced by Claude



Everything!!! done using Claude

(except the use of MF62GIS to write mif/mid files. But I could have asked Claude to do this.)

But I needed to know what to ask for.

Therefore, I needed to know something about MF6, and its input/output file protocols







The world has changed

Working with the Sagehen Model

(example supplied with MODFLOW 6)

Documents that I asked Claude to read

 gwutil_a.pdf	23/04/2025 6:55 AM	Microsoft Edge P...	569 KB
 gwutil_b.pdf	23/04/2025 6:56 AM	Microsoft Edge P...	3,070 KB
 gwutil_c.pdf	28/11/2025 2:58 PM	Microsoft Edge P...	1,963 KB
 mf6io.pdf	6/02/2026 12:25 AM	Microsoft Edge P...	3,444 KB

Getting help to run MF6PTINGRID

Conversation with Claude

me

There is a file named `points.dat` in this folder. This contains the coordinates of a set of points. What utility from the groundwater data utilities suite (i.e. the manuals that you have just read) is best for finding out what model cells these points are in?

Claude

`mf6ptingrid` — it's designed exactly for this. It reads a bore coordinates file and the MODFLOW6 binary grid file (`.grb`), and identifies which model cell each point falls in.

Your `points.dat` is already in the right format (name, x, y, layer), and you have `ex-gwf-sagehen.dis.grb` ready to go.

Would you like me to help you run it?

Getting help to run MF6PTINGRID

Claude told me

I have everything I need. Here's exactly how to run it. Open a command prompt in your `sagehen` folder and type:

```
mf6ptingrid
```

Then respond to each prompt as follows:

```
Enter name of MODFLOW6 binary grid file: ex-gwf-sagehen.dis.grb
```

```
Enter name of bore coordinates file: points.dat
```

```
Enter name of bore listing file: points.dat
```

```
Enter name for output file: points_cells.txt
```

For the bore listing file prompt, just supply `points.dat` again — that tells `mf6ptingrid` to process all points in the file.

The output file `points_cells.txt` will show the row and column of each point, plus its IDOMAIN value, cell top and bottom elevations. Note that because the `sagehen` DIS file has `XORIGIN`, `YORIGIN`, and `ANGROT` set, `mf6ptingrid` will correctly handle the rotated grid and your global coordinates in `points.dat`.

Getting help to run MF6PTINGRID

me

Can you place those responses into a file named mf6ptingrid.in so that I can run it using the "<" character for keyboard redirection to its input?

Created a file

Done. Run it from the sagehen

Claude

mf6ptingrid < mf6ptingrid.in

Running MF6PTINGRID

```
E:\myclaude\modflow6\MODFLOW6\sagehen>mf6ptingrid < mf6ptingrid.in
```

```
Program MF6PTINGRID evaluates the cell in which a point lies in a MODFLOW 6  
DIS or DISV grid.
```

```
Enter name of MODFLOW6 binary grid file: - file ex-gwf-sagehen.dis.grb read ok.
```

```
Enter name of bore coordinates file: - 7 bores and coordinates read from bore coordinates file points.dat
```

```
Enter name of bore listing file: - 7 bores read from bore listing file points.dat
```

```
Enter name for output file: - file points_cells.txt written ok.
```

POINT	EASTING	NORTHING	ICOL	IROW	ILAY	ICELLTYPE	IDOMAIN	TOP	BOTTOM
PT1	46205.862	770317.079	53	17	1	1	1	2135.000	2030.000
PT2	48335.114	769703.854	69	36	1	1	1	2018.000	1910.000
PT3	49016.475	767727.907	62	58	1	1	1	2121.000	2020.000
PT4	46870.188	766774.002	37	53	1	1	1	2148.000	2050.000
PT5	45303.058	767404.261	26	38	1	1	1	2259.000	2160.000
PT6	47040.529	768903.255	52	35	1	1	1	2003.000	1900.000
PT7	44332.119	766143.743	10	43	1	1	1	2420.000	2320.000

Displaying calculated heads in a GIS

me

Are you able to tell me the simulation times at which heads are available? Is there a documented utility program that will do this?

Yes — **mf6arrdet** will list the contents of the binary head file, including all simulation times (TOTIM values).

Claude

However, I can also read it directly since I know the binary format from CLAUDE.md. Would you prefer I just tell you the times now, or would you like to also try running **mf6arrdet** as a useful exercise?

me

Yes please, tell me what is in that file.

Ran a command

Claude

399 head arrays, one per stress period, TOTIM = 1 through 399 days — so this is a daily transient simulation running for 399 days (just over 13 months). Each stress period is 1 day long with 1 time step.



Claude doesn't need any help from my utility programs

Displaying calculated heads in a GIS

me

Did you write a little python script and run it on your server?

Yes — I read the binary `.hds` file directly using Python in my sandbox (which has access to your files via the mounted folder). The binary format is documented in `CLAUDE.md` from our last session, so I knew exactly how to parse the 52-byte headers and skip over the data arrays to extract the `KSTP`, `KPER`, `PERTIM`, `TOTIM` values.

Claude

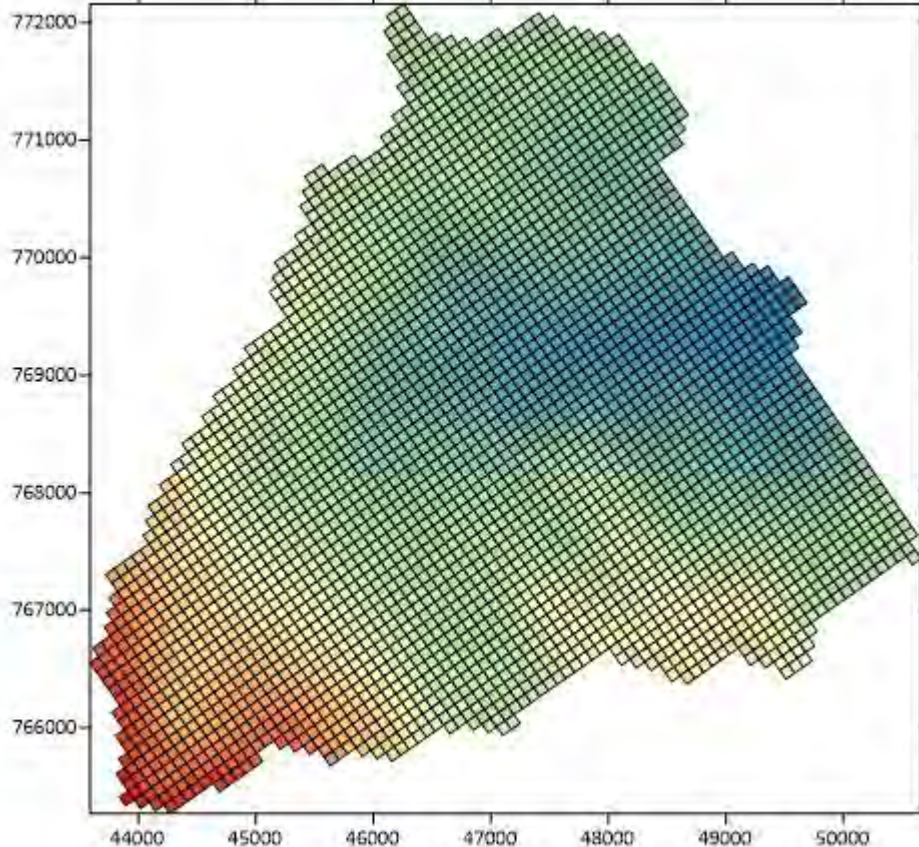
No need to run `mf6arrdet` for this one — I could do it faster myself.

I asked Claude

Displaying calculated heads in a GIS

Claude said

I would like to import the model into a GIS together with the heads at 10 days into the simulation. Is there a utility (or utilities that can do this)? Or can you do this more quickly yourself?



Two gwutil programs working together could do it:

1. **mf62gis** — reads the `.grb` and writes MIF/MID files of the grid geometry (for QGIS/MapInfo) or a BLN file (for Surfer)
2. **mf6dep2csv** or **mf6bin2tab_h** — extracts heads at TOTIM=10 as a tabular file

You'd then join the head values to the grid polygons inside your GIS — an extra manual step.

I can do it more directly: read the `.grb` geometry and `.hds` heads at day 10 myself, apply the 35° rotation to compute proper global coordinates for each cell, and write a single GeoJSON or shapefile with the head values already attached. You'd just drag it into your GIS.

Which GIS are you using, and which output format would suit you best — GeoJSON, shapefile, or something else?

I'm using qgis. Shapefile would be fine.

Ran 5 commands

Done. Four files are in your `sagehen` folder: `sagehen_heads_day10.shp/.shx/.dbf/.prj`.

In QGIS, go to Layer → Add Layer → Add Vector Layer and open the `.shp` file. Each cell is a polygon with a `HEAD` attribute you can use for symbology. The 35° grid rotation is correctly applied so it will sit in the right place.

One caveat: I've assumed **UTM Zone 10N (WGS84)** for the coordinate system in the `.prj` file, as Sagehen Creek is in the Sierra Nevada of California. If the model uses a different CRS, let me know and I'll update the `.prj`.

Displaying calculated heads in a GIS

I asked Claude

That worked well. I would like to be able to produce a similar GIS-ready set of files whenever I run the model. Are you able to write me a tool to do this? Is this what you call an artifact?

Claude said

Thought process

Not quite — in Cowork, an "artifact" is a live interactive HTML dashboard. What you want is simpler and more useful: a Python script you can run any time after a model run, similar to `make_vtk.py` and `make_map.py` we wrote for mymodel.

Let me write a proper version that accepts the simulation time as a command-line argument, so you can do things like `python make_shp.py 10` or `python make_shp.py 365`.

Ran 4 commands, created 2 files, read 2 files

The file was truncated during writing. Let me write it via the bash sandbox instead.

Ran 2 commands

Working — run it from any MODFLOW6 model folder with:

```
python make_shp.py 10
```

or just `python make_shp.py` to list the available times first. It auto-detects the model name, handles grid rotation, and names output files with zero-padded TOTIM so they sort cleanly (`_t0010`, `_t0365`, etc.).

Displaying calculated heads in a GIS

Running the Python script that Claude wrote for me

```
E:>python make_shp.py
Model prefix : ex-gwf-sagehen
Grid         : 73 rows x 81 cols, cell 90.0 x 90.0 m
Origin       : (45235.0, 763422.0), ANGROT=35.0 deg
Active cells : 3387
```

399 head arrays in ex-gwf-sagehen.hds:

KSTP	KPER	TOTIM
1	1	1.000
1	2	2.000
1	3	3.000
1	4	4.000
1	5	5.000
1	6	6.000
...		
1	387	387.000
1	388	388.000
1	389	389.000
1	390	390.000
1	391	391.000
1	392	392.000
1	393	393.000
1	394	394.000
1	395	395.000
1	396	396.000
1	397	397.000
1	398	398.000
1	399	399.000

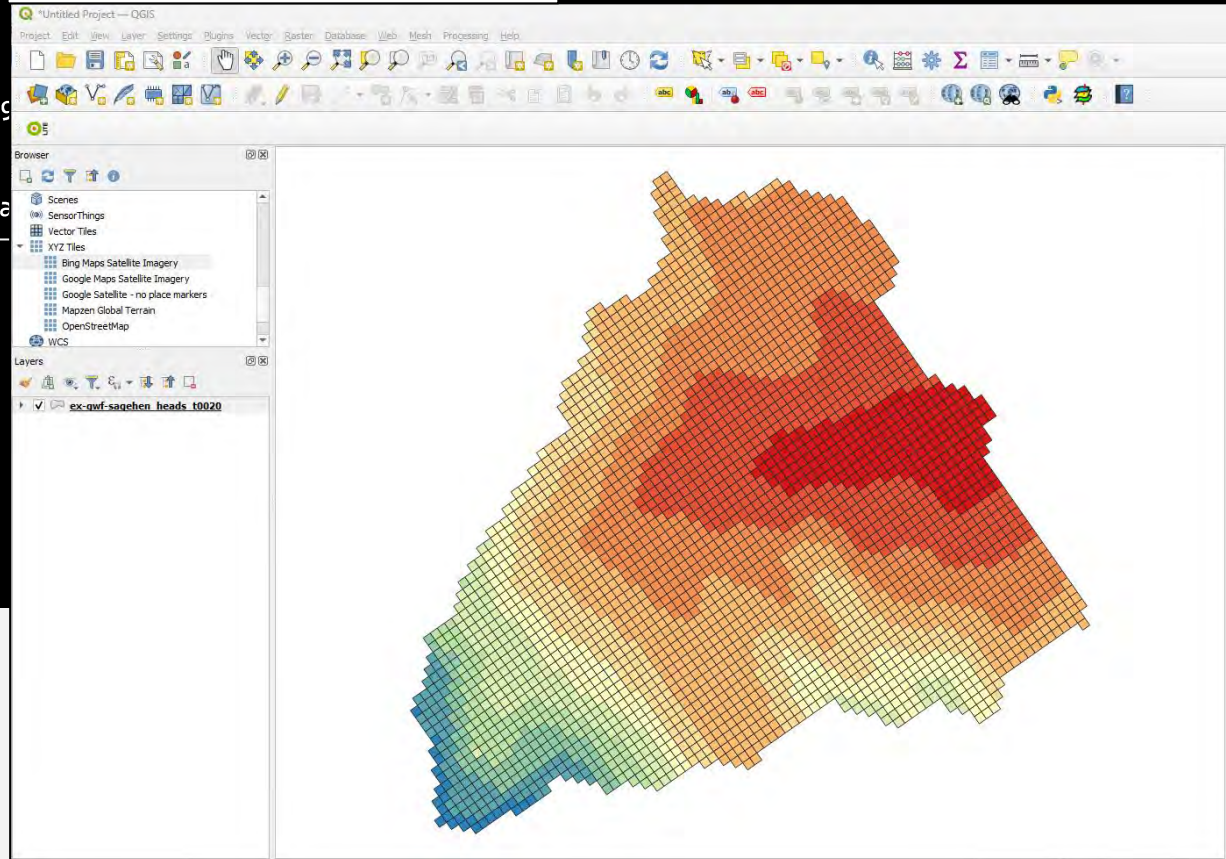
Usage: python make_shp.py <totim>

Running the Python script that Claude wrote for me

```
E:\myclaude\modflow6\MODFLOW6\sagehen>E:\temp_copy_myclause\modflow6\MODFLOW6\python make_shp.py 398
Model prefix : ex-gwf-sagehen
Grid         : 73 rows x 81 cols, cell 90.0 x 90.0 m
Origin       : (45235.0, 763422.0), ANGROT=35.0 deg
Active cells : 3387
```

```
Reading heads at TOTIM=398
Writing 3387 polygons to
Done.
Open ex-gwf-sagehen_heads
```

Importing the shapefile into QGIS



Displaying calculated heads in a GIS

I asked Claude

can you write that some program in FORTRAN? I can then compile it using IFORT.

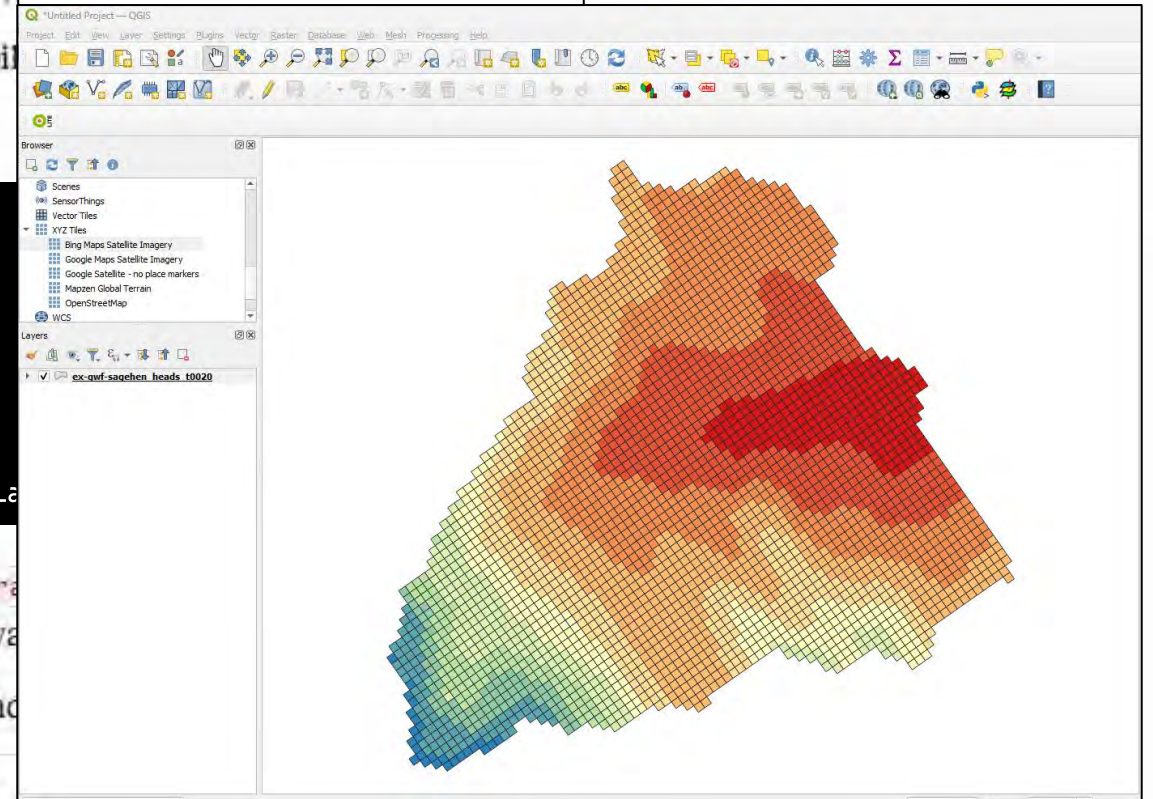
Claude said

Running the FORTRAN program that Claude wrote for me

```
E:\myclaude\modflow6\MODFLOW6\sagehen>make_shp 20 ex-gwf-sagehen
Model prefix : ex-gwf-sagehen
Grid       : 73 rows x 81 cols, cell    90.0 x    90.0 m
Origin    : ( 45235.0, 763422.0)  ANGROT= 35.0 deg
Active cells : 3387

Reading heads at TOTIM=20.000 ...
Writing 3387 polygons to ex-gwf-sagehen_heads_t0020.* ...
Done.
Open ex-gwf-sagehen_heads_t0020.shp in QGIS via Layer > Add Layer > Add Vector La
```

Importing the shapefile into QGIS



Bug fixed. The read_array
using a separate ngot va

To compile on your Wind

```
ifx -o make_shp make_shp.190
```

Note: there are no shapefile libraries that I know of that are easily accessible from FORTRAN. So Claude wrote the shapefile and database access software itself.

Basic Uncertainty Analysis

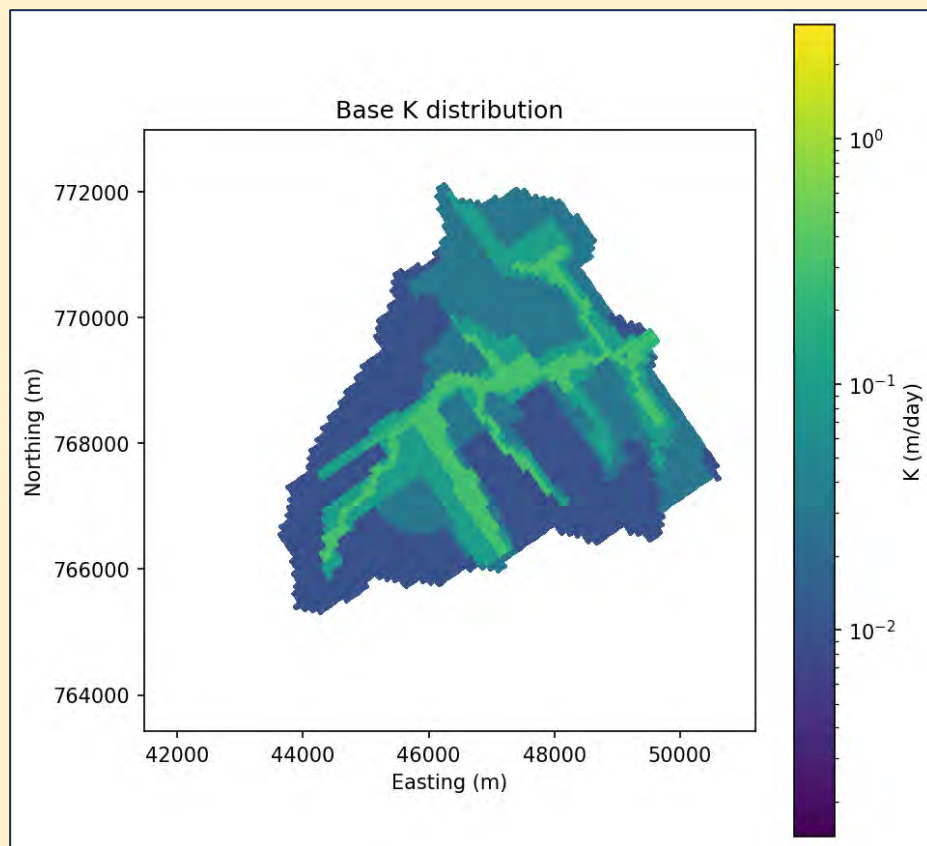
K field used by the Sagehen model

I asked Claude

Generate a picture of the model's K field.

Claude said

..the existing model K field isn't uniform — it has 7 distinct values (0.005–0.3 m/day) representing different geologic zones...



Claude generated a *png* file using a Python script running in its bash sandbox. (It is happy to share any script that it writes with me.)

Stochastic field generation

What I asked Claude to do

- Generate stochastic K-multiplier fields for the Sagehen model. (It could do this using Python or FIELDGEN+TWOARRAY.) I preferred the latter option as I wanted to place the field generation and multiplication process in a batch loop which could be transferred to computers that do not have Python installed.
- Write a batch file that will run the model multiple times using these K fields.
- Store the unformatted heads arrays in name-indexed files
- Calculate the difference between stochastic head fields and the base field at the last time step of the model. (Claude used Python for this.)
- Make some pictures (Claude used Python for this.)
- Then make a slideshow of what we did, and any tricks learned along the way about using the Groundwater Utility suite.

Why bother about the Groundwater Utility Suite?

Good question. Some of them will be useful later in this slideshow for specialist tasks such as undertaking nonstationary spatial interpolation and stochastic field generation for which Python options do not exist (as far as I know).

Here is Claude's slideshow

Stochastic Groundwater Modelling

with Claude & MODFLOW 6

Sagehen Creek Watershed · Stochastic K Fields · Ensemble Heads

A workflow developed collaboratively between a hydrogeologist and Claude

Using: MODFLOW 6 · Groundwater Utilities · Python / matplotlib

What We Built Together

1 Stochastic K fields

Used FIELDGEN (Groundwater Utilities) to generate 10 random hydraulic conductivity multiplier fields for the Sagehen Creek model domain.

2 Field combination

Used TWOARRAY to multiply each stochastic multiplier against the base K field, producing 10 realisations of spatially variable hydraulic conductivity.

3 Ensemble runs

A Windows batch loop ran MODFLOW 6 for each realisation, storing the resulting heads file and K field in the stoch\ folder.

4 Visualisation

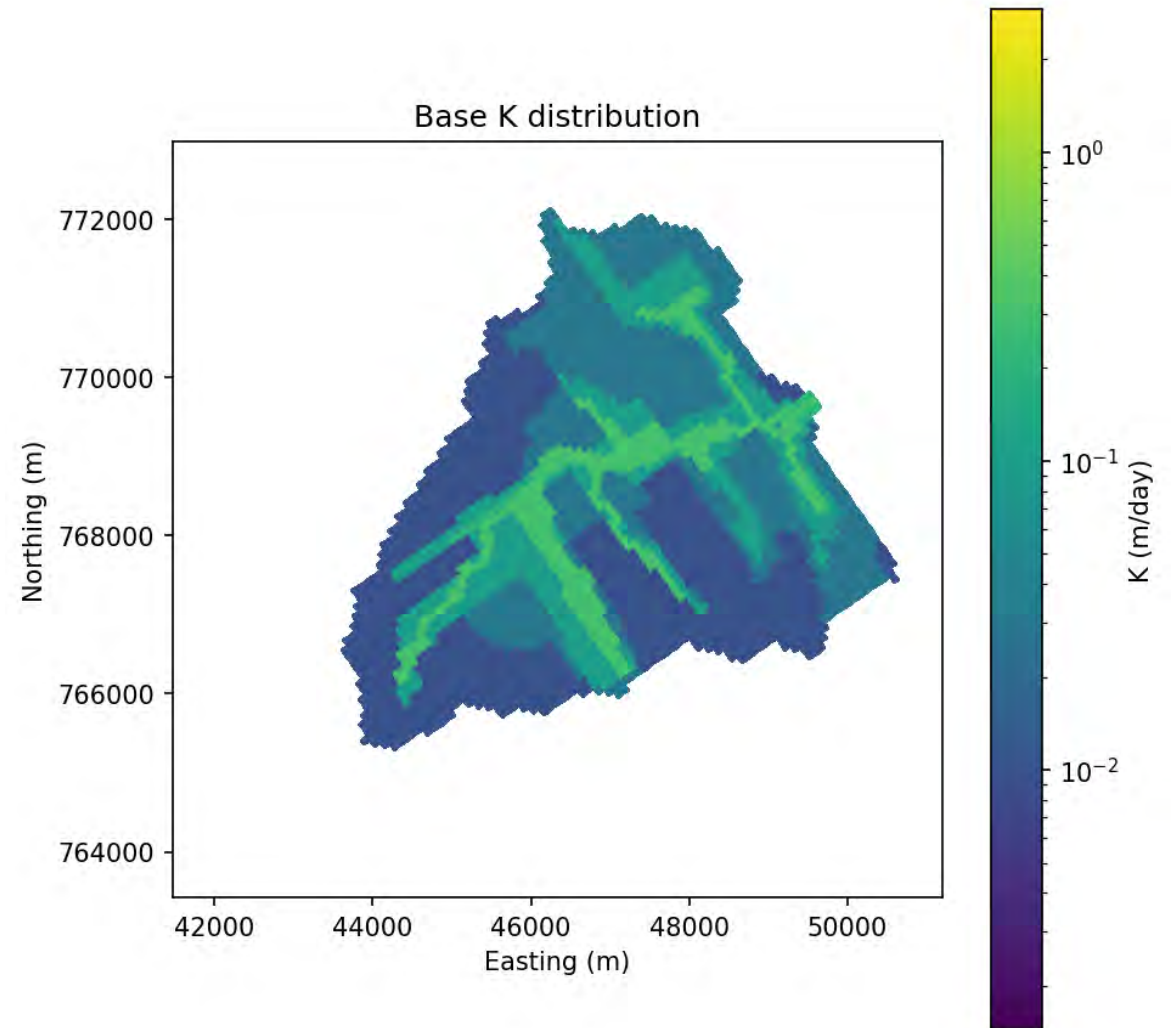
Python/matplotlib scripts produced maps of $\log_{10}(K)$ and head differences (stochastic minus base) for visual comparison across realisations.

The Sagehen Creek Model

MODFLOW 6 Example Model

Grid	73 rows × 81 columns, 5,913 cells
Cell size	90 m × 90 m (uniform)
Watershed	Sagehen Creek, Sierra Nevada, California
Model type	Single layer, transient groundwater flow
K field	Spatially variable hydraulic conductivity
Packages	DIS, NPF, CHD, DRN, RCHA, OC, IMS, TDIS

Base Hydraulic Conductivity (K)



Why Stochastic Modelling?

Hydraulic conductivity (K) is the most uncertain parameter in groundwater modelling. Stochastic methods propagate this uncertainty through to model predictions.

Uncertainty in K

K varies over many orders of magnitude across a watershed and cannot be measured everywhere. Any single K field is therefore just one possible realisation.

Ensemble approach

By running many realisations — each with a different but plausible K field — we build a distribution of predicted heads rather than a single deterministic answer.

Multiplier fields

A stochastic multiplier field is generated from a log-normal distribution and multiplied against the base K. This preserves spatial structure while introducing realistic variability.

Uncertainty in heads

Comparing head differences across realisations shows which parts of the model domain are most sensitive to K uncertainty — useful for guiding further data collection.

Groundwater Utilities (gwutil)

A suite of standalone DOS/Windows programs for pre- and post-processing MODFLOW-compatible arrays. Each program reads input from keyboard (or a redirected file) and writes results to disk.

How they work — keyboard redirection

```
program < responses.in
```

Each program prompts for inputs in sequence. By redirecting a text file of answers, the entire interaction is automated — no manual typing required.

FIELDGEN

Generate spatially correlated random fields (log-normal). Used to create stochastic K multiplier realisations.

TWOARRAY

Combine two real arrays by addition, subtraction, multiplication, or division, cell by cell.

REAL2SRF

Convert a real array to SURFER grid format for visualisation.

MF62VTK2

Convert MODFLOW 6 binary output to VTK format for 3D visualisation in Paraview.

MF6DEP2CSV

Extract dependent variables (heads, concentrations) to CSV for spreadsheet analysis.

FIELDGEN — Generating Stochastic Multiplier Fields

FIELDGEN produces spatially correlated random fields on a MODFLOW grid. We used it to generate 10 log-normal multiplier fields — each a plausible pattern of K variability.

fieldgen.in — redirect input file

```
ex-gwf-sagehen.spc
10
sagehen_zones.inf
1
0.0
1.0
0.3
500.0
500.0
0.0
kreal
0
```

What each line means

- Grid specification file
- Number of realisations (10)
- Zone integer array file (.inf extension → no format prompt)
- Zone number to generate within
- Mean of $\log_{10}(K)$ multiplier (0 = multiplier centred on 1)
- Standard deviation of $\log_{10}(K)$
- Correlation length (fraction of domain)
- Correlation range — X direction (m)
- Correlation range — Y direction (m)
- Anisotropy angle (degrees)
- Output filename prefix → kreal1.ref ... kreal10.ref
- Random seed (0 = use system clock)

Key Tip for Claude Users: File Extensions

This is the single most important thing to tell Claude before working with Groundwater Utilities:

.ref Real array files

Formatted real (floating-point) arrays

Examples:

k_base.ref
k_stoch.ref
kreal1.ref

Why it matters:

gwutil programs detect .ref and automatically use formatted real arrays — no format prompt is shown.

.inf Integer array files

Formatted integer arrays

Examples:

sagehen_zones.inf
idomain.inf

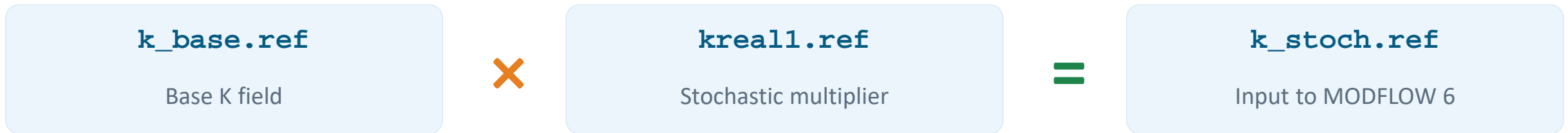
Why it matters:

gwutil programs detect .inf and automatically use formatted integer arrays — no format prompt is shown.

Without these extensions, gwutil programs ask 'Write a formatted or unformatted file? [f/u]:' — and if Claude's input file does not answer this extra prompt, the program loops indefinitely.

TWOARRAY — Combining the K Fields

TWOARRAY multiplies the base K field by each stochastic multiplier realisation, cell by cell, producing the input K field for each MODFLOW run.



twoarray.in — redirect input file

```
ex-gwf-sagehen.spc      ← grid spec
k_base.ref              ← first array (.ref = no prompt)
(blank)                 ← blank = no inactive threshold
kreal1.ref              ← second array (.ref = no prompt)
(blank)                 ← blank = no inactive threshold
m                       ← operation: multiply
k_stoch.ref             ← output (.ref = no prompt)
```

Key points

- Output file `k_stoch.ref` uses the `.ref` extension — so no format prompt appears
- The NPF package references `k_stoch.ref` via OPEN/CLOSE — MODFLOW reads it fresh each run
- 10 different `kreal*.ref` files are used across the 10 realisations
- Other operations available: replace (r), add (a), subtract (s), divide (d)

The Batch Workflow — run_stoch.bat

A Windows batch file loops over all 10 realisations, running TWOARRAY and saving results:

```
@echo off
setlocal enabledelayedexpansion
mkdir stoch 2>nul

for /L %%i in (1,1,10) do (
    set num=0%%i
    set num=!num:~-2!

    (echo ex-gwf-sagehen.spc
    echo k_base.ref
    echo.
    echo kreal%%i.ref
    echo.
    echo m
    echo k_stoch.ref) > twoarray.in

    twoarray < twoarray.in
    copy k_stoch.ref stoch\k_!num!.ref
)
```

— Silent mode; delayed expansion for ! variables inside the loop

— Zero-pad counter: 01, 02 ... 10

— Loop from 1 to 10

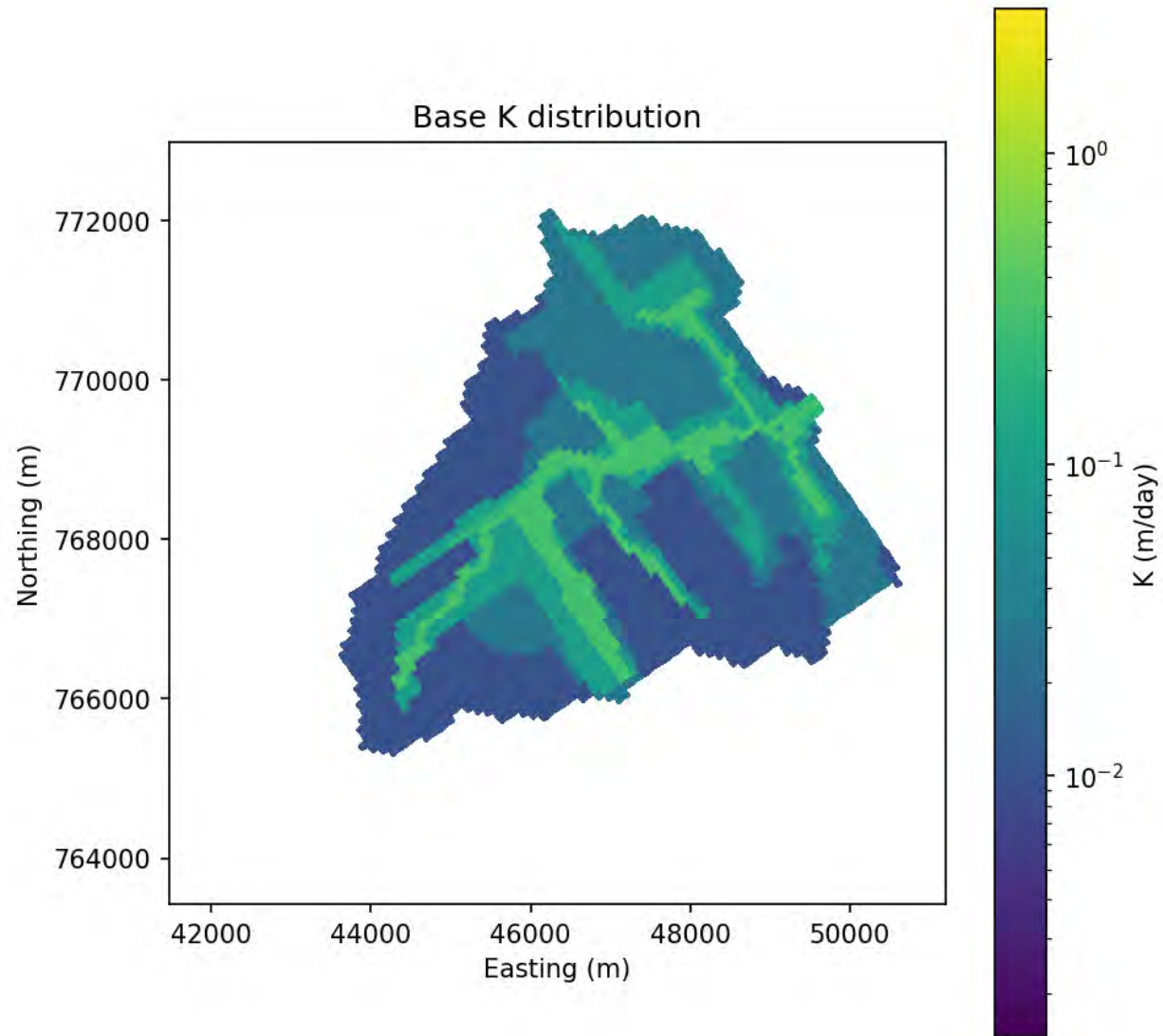
— Build the twoarray redirect file on the fly inside the loop

— Note: kreal%%i.ref not kreal0%%i.ref — FIELDGEN uses no zero-padding

— Run twoarray, then save the K realisation

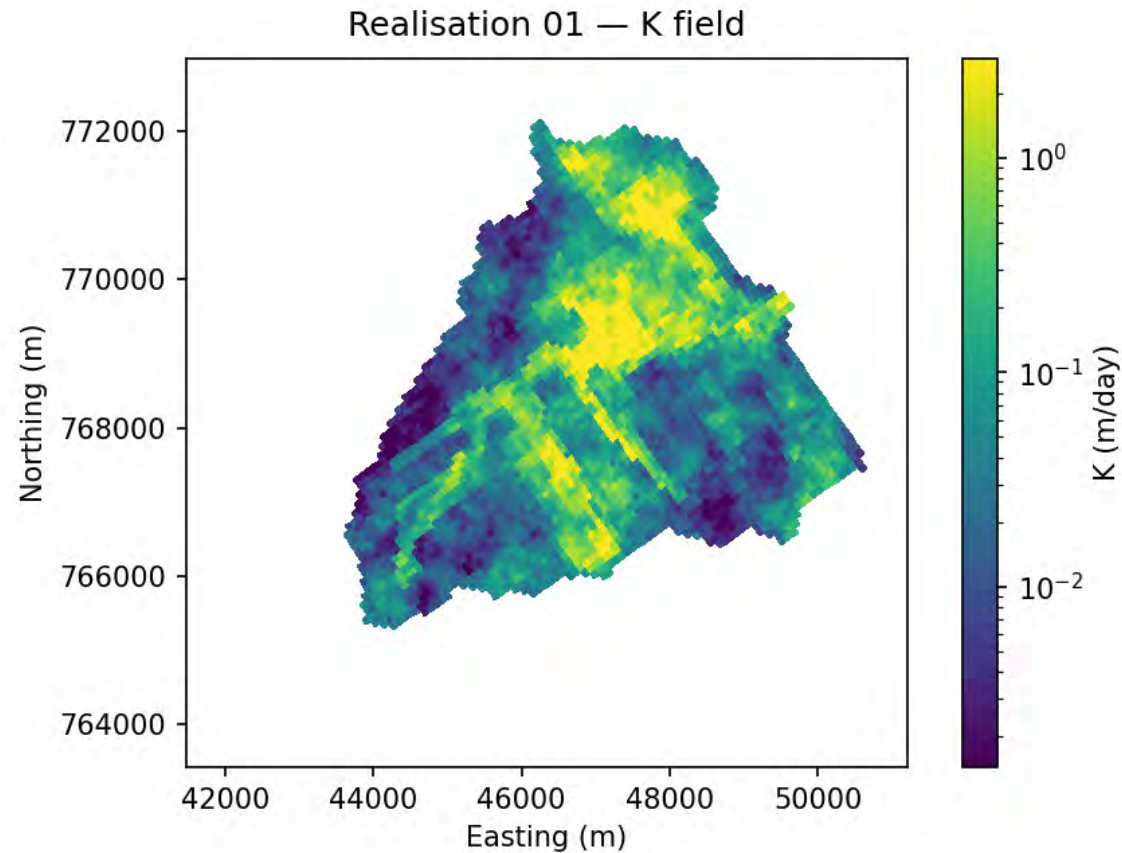
Outputs: stoch\k_01.ref ... k_10.ref paired with hds_01.hds ... hds_10.hds

Base Hydraulic Conductivity Field

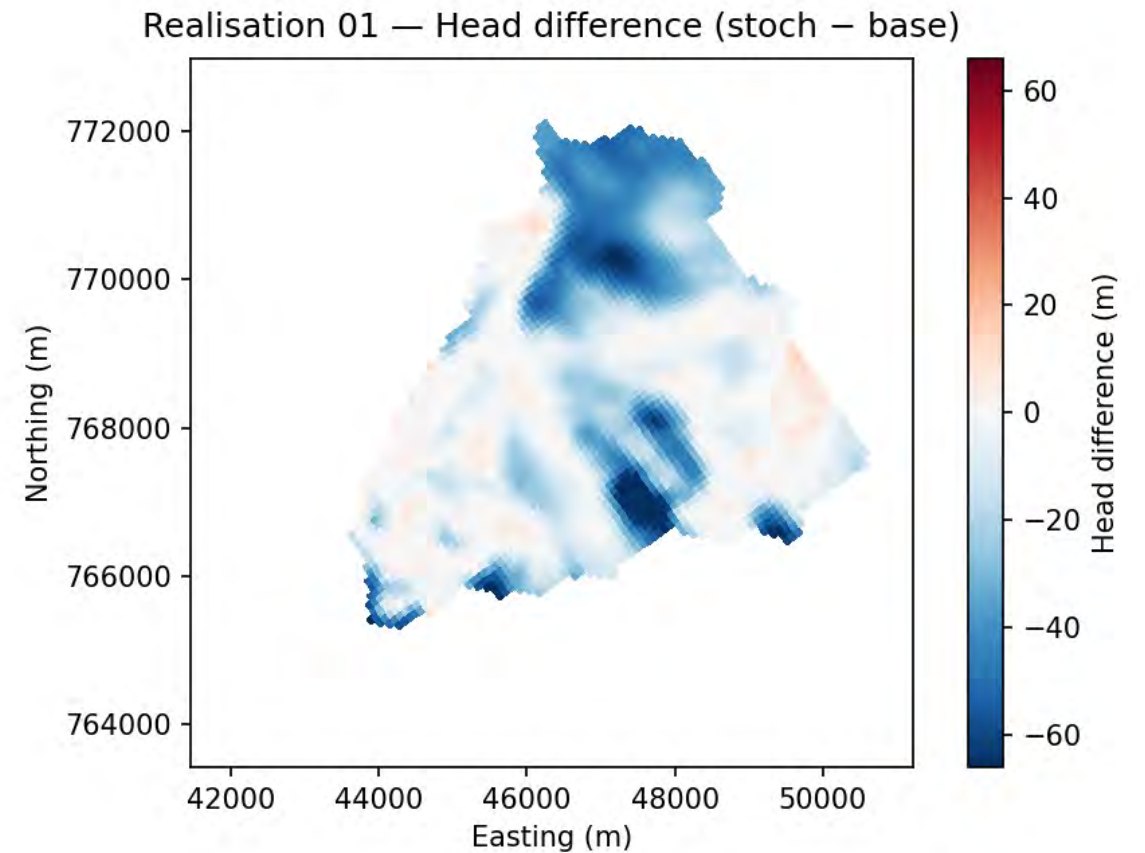


The deterministic K field used in the base MODFLOW 6 model. $\log_{10}(K)$ shown, viridis colour scale. This is the starting point for the stochastic analysis.

Realisation 01 — K field and Head Difference



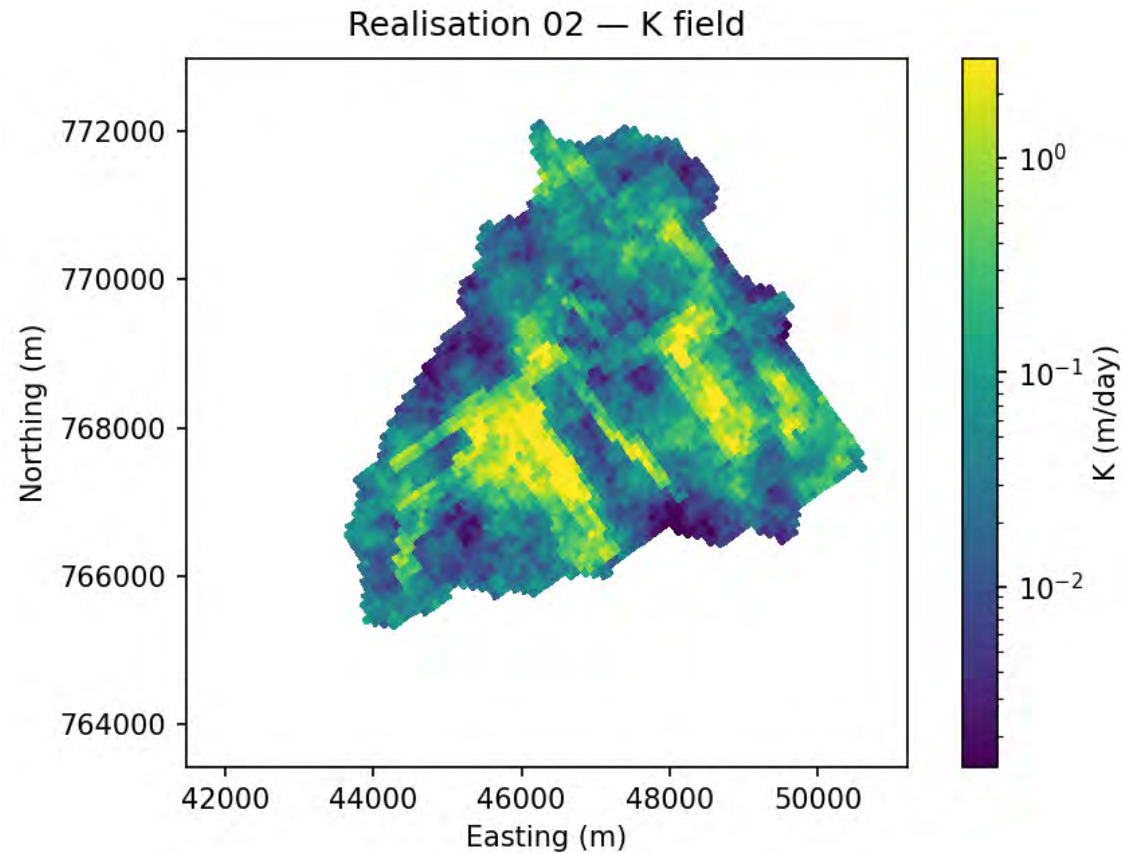
Left: $\log_{10}(K)$ — viridis, consistent scale across all realisations



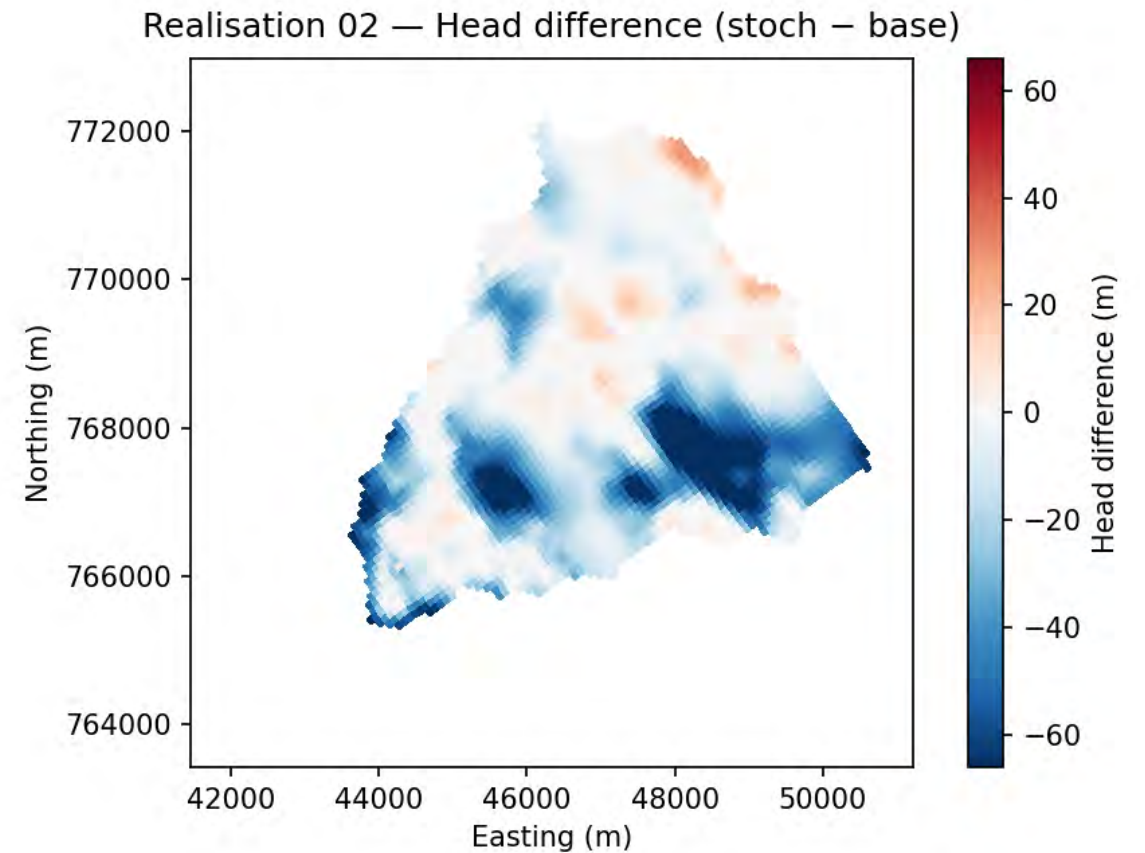
Right: head difference (stochastic - base, m) — red/blue diverging

Realisation 01 of 10. Each uses a different FIELDGEN random seed. Colour scales are consistent across all realisations for direct comparison.

Realisation 02 — K field and Head Difference



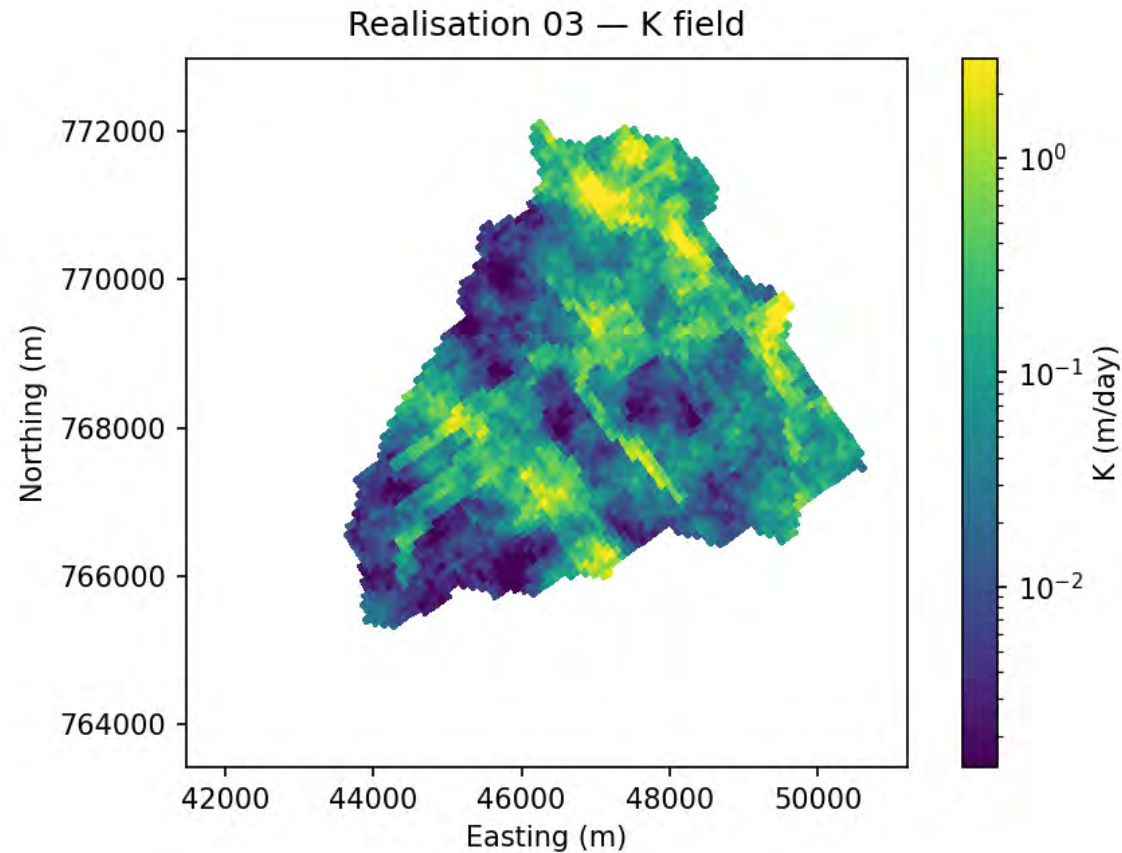
Left: $\log_{10}(K)$ — viridis, consistent scale across all realisations



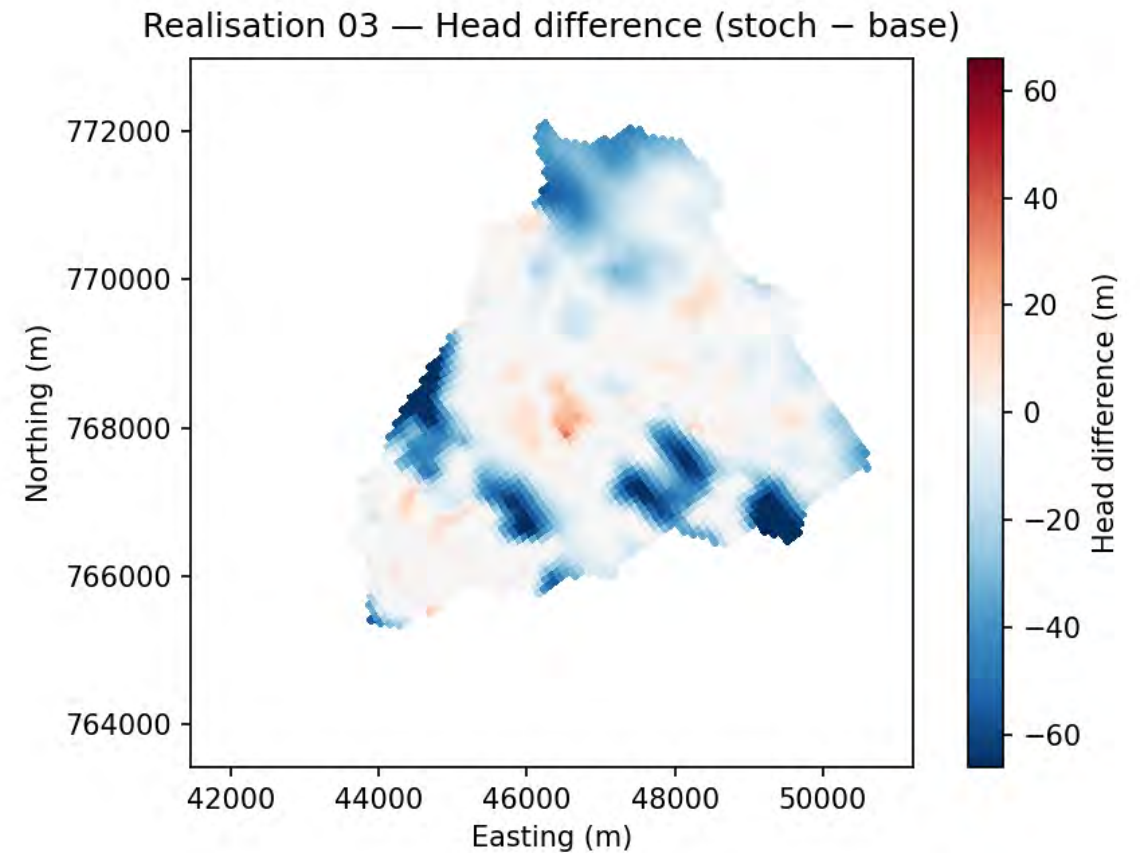
Right: head difference (stochastic – base, m) — red/blue diverging

Realisation 02 of 10. Each uses a different FIELDGEN random seed. Colour scales are consistent across all realisations for direct comparison.

Realisation 03 — K field and Head Difference



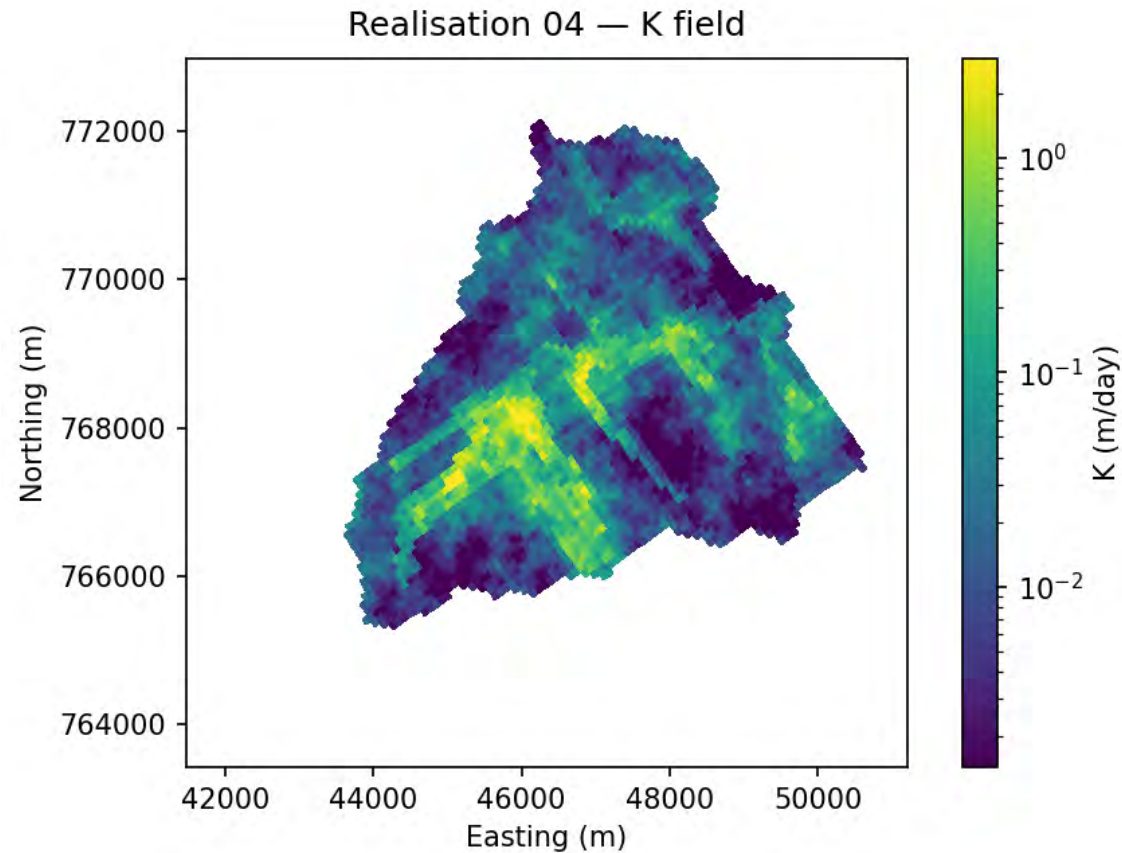
Left: $\log_{10}(K)$ — viridis, consistent scale across all realisations



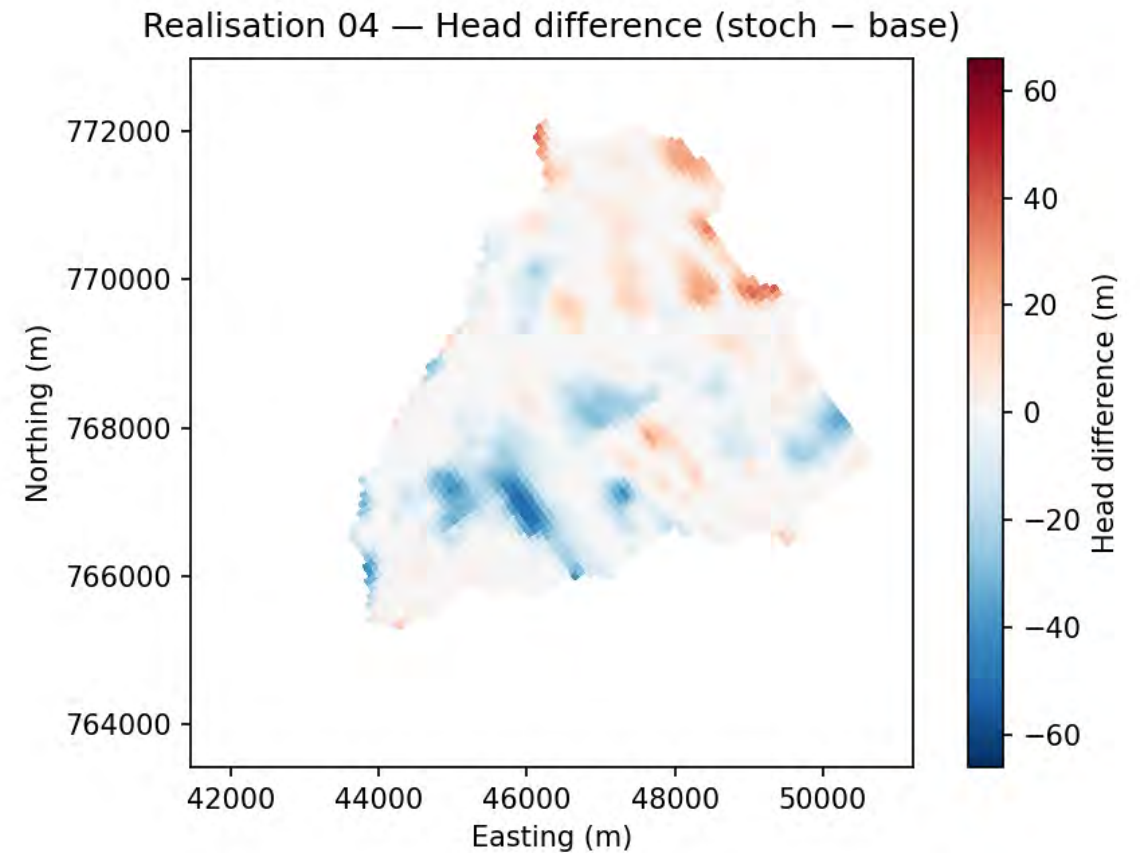
Right: head difference (stochastic – base, m) — red/blue diverging

Realisation 03 of 10. Each uses a different FIELDGEN random seed. Colour scales are consistent across all realisations for direct comparison.

Realisation 04 — K field and Head Difference



Left: $\log_{10}(K)$ — viridis, consistent scale across all realisations



Right: head difference (stochastic – base, m) — red/blue diverging

Realisation 04 of 10. Each uses a different FIELDGEN random seed. Colour scales are consistent across all realisations for direct comparison.

Claude's Python Capabilities

Beyond running command-line utilities, Claude can write and execute Python scripts directly — reading MODFLOW binary output, computing statistics, and producing publication-quality figures.

What Claude did in this project

- **Parsed the DIS file**
Extracted grid origin, rotation, cell sizes, and IDOMAIN to build accurate UTM coordinate arrays
- **Read .ref files**
Parsed plain-text real array files into NumPy arrays (NROW × NCOL)
- **Read binary .hds files**
Unpacked MODFLOW 6 binary head output using struct — no external library needed
- **Applied masking**
Masked inactive cells (IDOMAIN ≠ 1) and HNOFLO values for clean maps
- **Plotted pcolormesh**
Used matplotlib pcolormesh with corner coordinates for geographically correct plots
- **Consistent colour limits**
Computed global limits across all realisations so maps can be compared directly

What else is possible

- Water balance analysis from the MODFLOW list file
- Time-series extraction at observation points
- Statistical summaries of ensemble heads (mean, std dev, percentiles)
- Breakthrough curve analysis for transport models
- Particle tracking visualisation (MODPATH output)
- Automated model calibration reporting
- Sensitivity analysis plots across parameter ranges

Key Tips for Your Claude Installation

Tell Claude these things at the start of any Groundwater Utilities session:

1 The .ref / .inf convention

Real arrays → .ref, integer arrays → .inf. gwutil detects these extensions automatically. Never add format prompts to redirect files.

3 Redirect files, not interaction

All gwutil programs work via: program < responses.in. Claude should always build the redirect file — never assume interactive input.

5 Python for visualisation

For reading .ref and .hds files and producing ensemble figures, ask Claude to write Python. It runs directly in Claude's sandbox with no extra tools.

2 FIELDGEN output naming

FIELDGEN uses no zero-padding: kreal1.ref ... kreal10.ref. Batch files must use the raw loop counter, not the zero-padded form.

4 OPEN/CLOSE in NPF

The MODFLOW 6 NPF package supports OPEN/CLOSE to read K from file. Update the NPF to reference k_stoch.ref before each batch run.

6 Save facts to CLAUDE.md

Ask Claude to save key conventions to CLAUDE.md in your project folder. Claude reads this file at the start of future sessions to restore context.

Building a Voronoi-Gridded Model

Working with Voronoi grids

First, I built a basic MODFLOW 6 input dataset with the correct geometry and number of layers using VOROGRIDGEN (downloadable from <https://hydrosymple.com/en/free-software/>)

VOROGRIDGEN input file

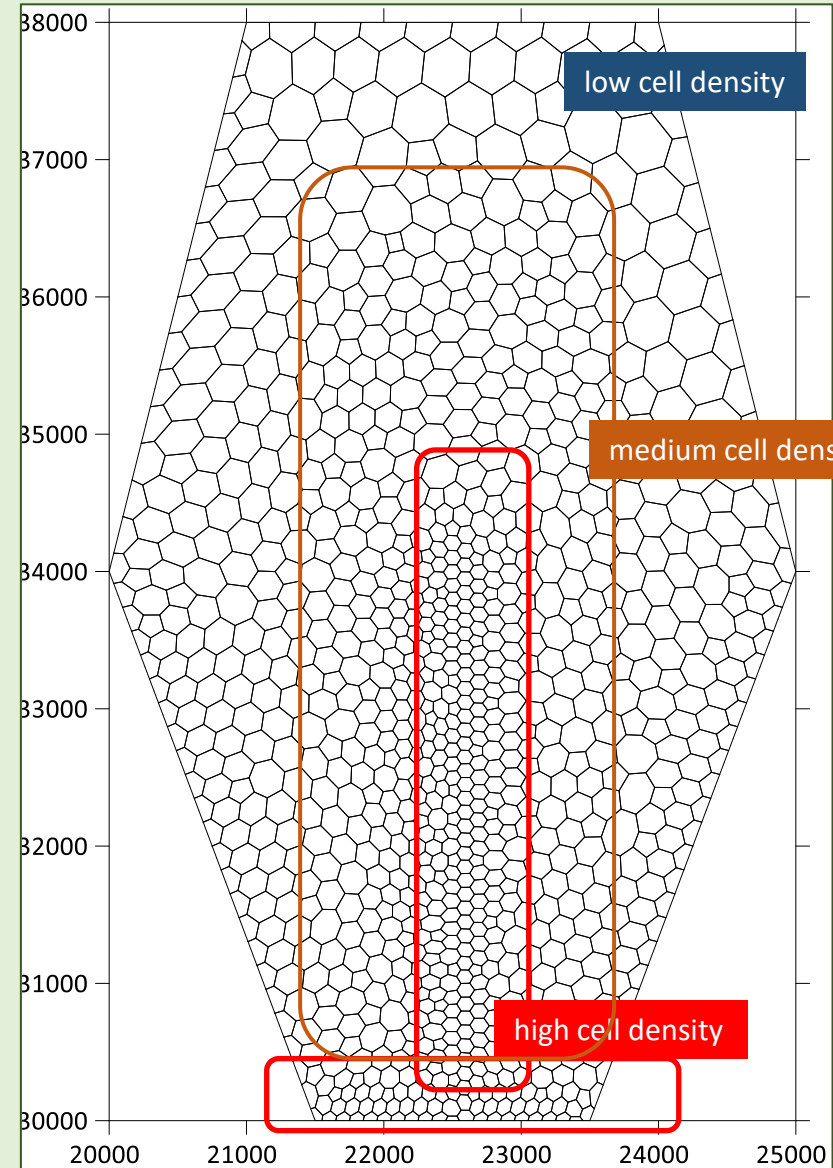
```
START OUTER_BOUNDARY
  bln_file=outer.bln
END OUTER_BOUNDARY

START INNER_LINE
  numlines=1
  bln_file=drain.bln
END INNER_LINE

START INNER_POLYGON
  bln_file=inner.bln
  max_centroid_separation=200.0
END INNER_POLYGON

START MF6
  xorigin=0.0
  yorigin=0.0
  nlay 3
  mf6_basename=model
END MF6

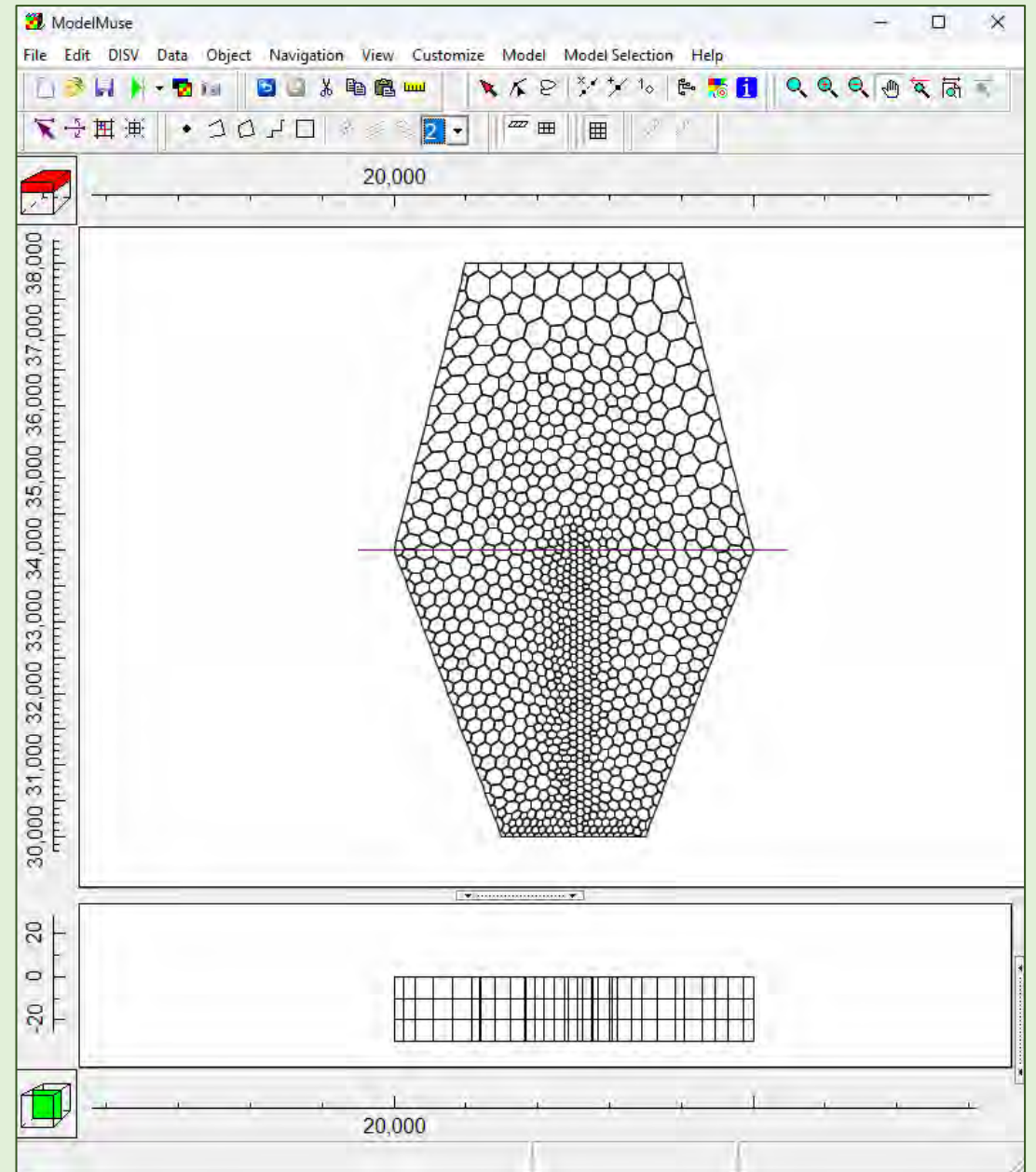
START CONTROL
  poly_growth_rate=1.2
  max_centroid_separation=550.0
  out_file_base=test
  max_cells=50000
  safety=0
  max_lloyd=30
  eps_lloyd=1.0d-5
  lloyd_fac=0.1
  nsdim=41
END_CONTROL
```



Working with Voronoi grids

Import the basic model into ModelMuse

- three layers (as requested)
- flat layer boundaries (this is what VOROGRIDGEN does)



Working with Voronoi grids – importing a DTM

File with surface elevations on a regular grid, i.e. a DTM (any format would do)

easting	northing	elevation
20005.000	37995.000	130.000000
20015.000	37995.000	130.000000
20025.000	37995.000	130.000000
20035.000	37995.000	130.000000
20045.000	37995.000	130.000000
20055.000	37995.000	130.000000
20065.000	37995.000	130.000000
20075.000	37995.000	130.000000
20085.000	37995.000	130.000000
20095.000	37995.000	130.000000
20105.000	37995.000	130.000000
20115.000	37995.000	130.000000
20125.000	37995.000	130.000000
20135.000	37995.000	130.000000
20145.000	37995.000	130.000000
20155.000	37995.000	130.000000
20165.000	37995.000	130.000000
20175.000	37995.000	130.000000
20185.000	37995.000	130.000000
20195.000	37995.000	130.000000
20205.000	37995.000	130.000000
20215.000	37995.000	130.000000
20225.000	37995.000	130.000000

...

Me

The present DISV file assigns constant values to the top of the model and to the bottom of model layers. I would like these top and bottom arrays accessed through external files using the OPEN/CLOSE protocol. I would like the top array informed by extrapolation from the contents of file *frame.xyz* in the *dtm* subfolder. I would like the bottom of layer 1 to slope evenly from an elevation of 90 metres in the north to 80 metres in the south of the model domain. I would like layer 2 to be 3 metres thick. I would like the bottom of layer 3 to have a uniform elevation of 60 metres. Can you do this?

Claude

- rewrote DISV file to access top/bottom elevation arrays using OPEN/CLOSE protocol
- interpolated from DTM to top of model
- wrote top/bottom arrays in text files

Me

Now build a VTK file of the model grid. What is best for you? To use one of the Groundwater Utilities or do it yourself?

Claude

Claude preferred to do it itself in its linux/Python sandbox

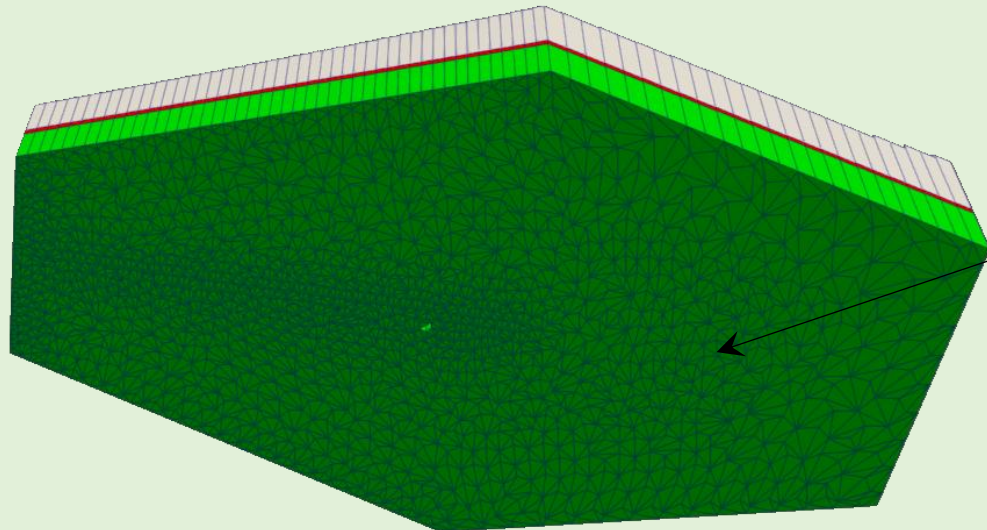
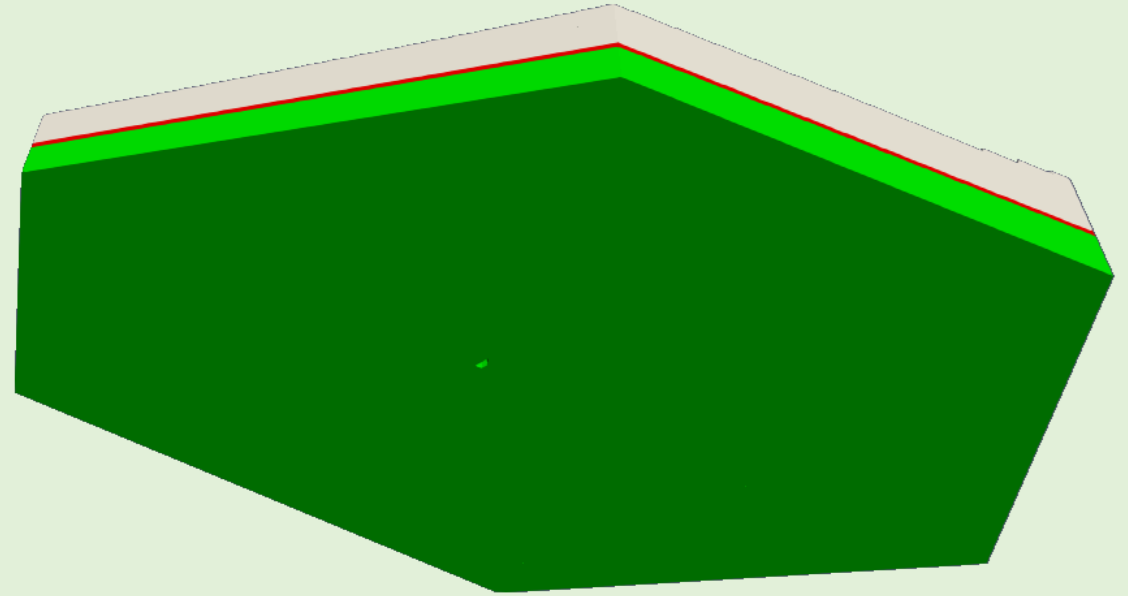
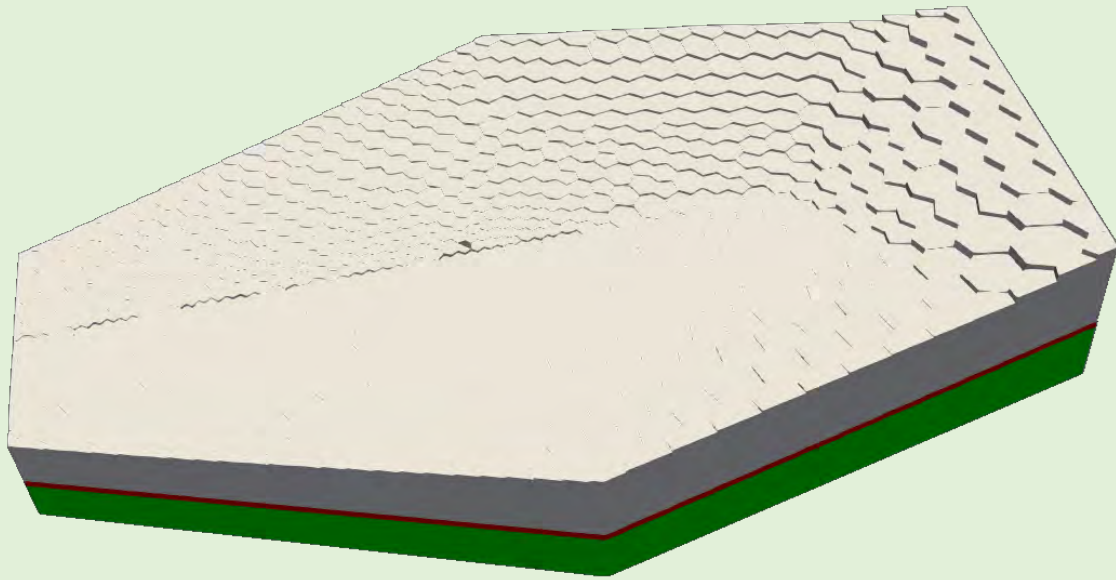
On inspecting the model in PARAVIEW, I discovered that Claude misread the DTM file and reversed the direction of the surface slope.

I asked it to correct this mistake.

It found the mistake and rebuilt the MODFLOW 6 top elevation array.

But...

Working with Voronoi grids

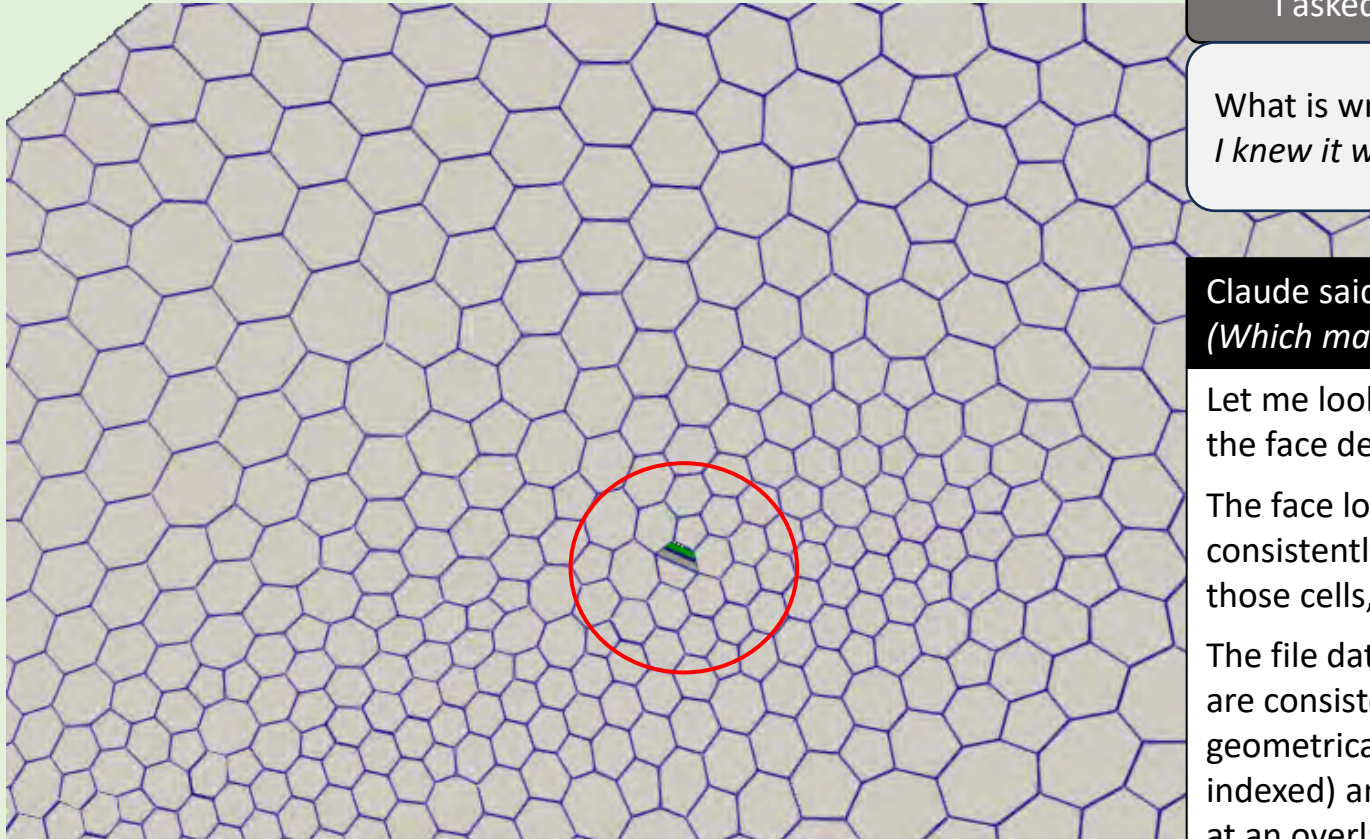


I asked Claude

Can you write the VTK file so that it displays actual model cells instead of this weird triangulation?

It did this, but..

Working with Voronoi grids



I asked Claude

What is wrong with cell 1009?
I knew it was cell 1009 through PARAVIEW.

Claude said it was my fault. This is what it said.
(Which made me worry that there may be a bug in VOROGRIDGEN.)

Let me look at what's actually in cell 1009 in the CELL2D data and check whether the face definitions are being constructed correctly for it.

The face logic looks correct — all point refs are in bounds and the winding is consistently clockwise. Let me look at the actual content written to the file around those cells, and check whether the issue might be in the cumulative face offsets:

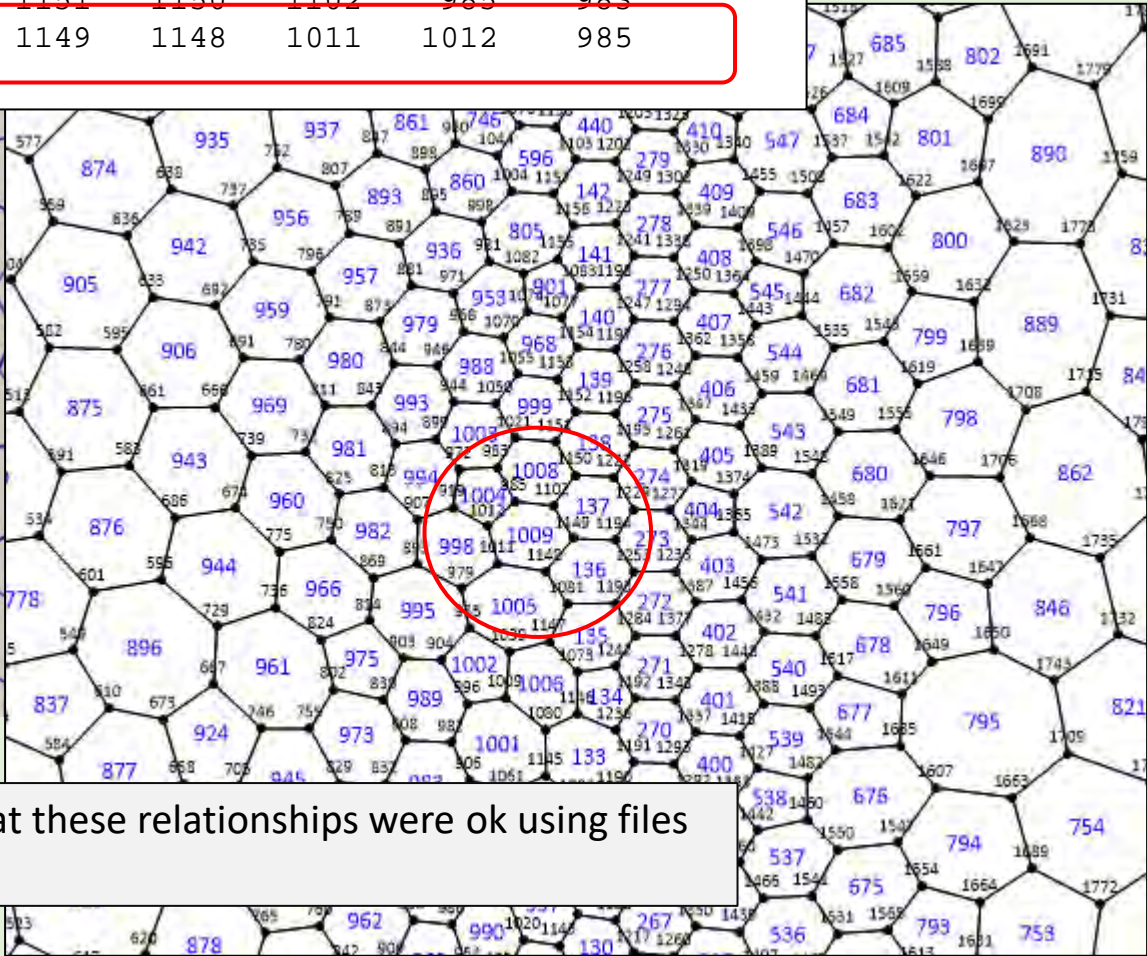
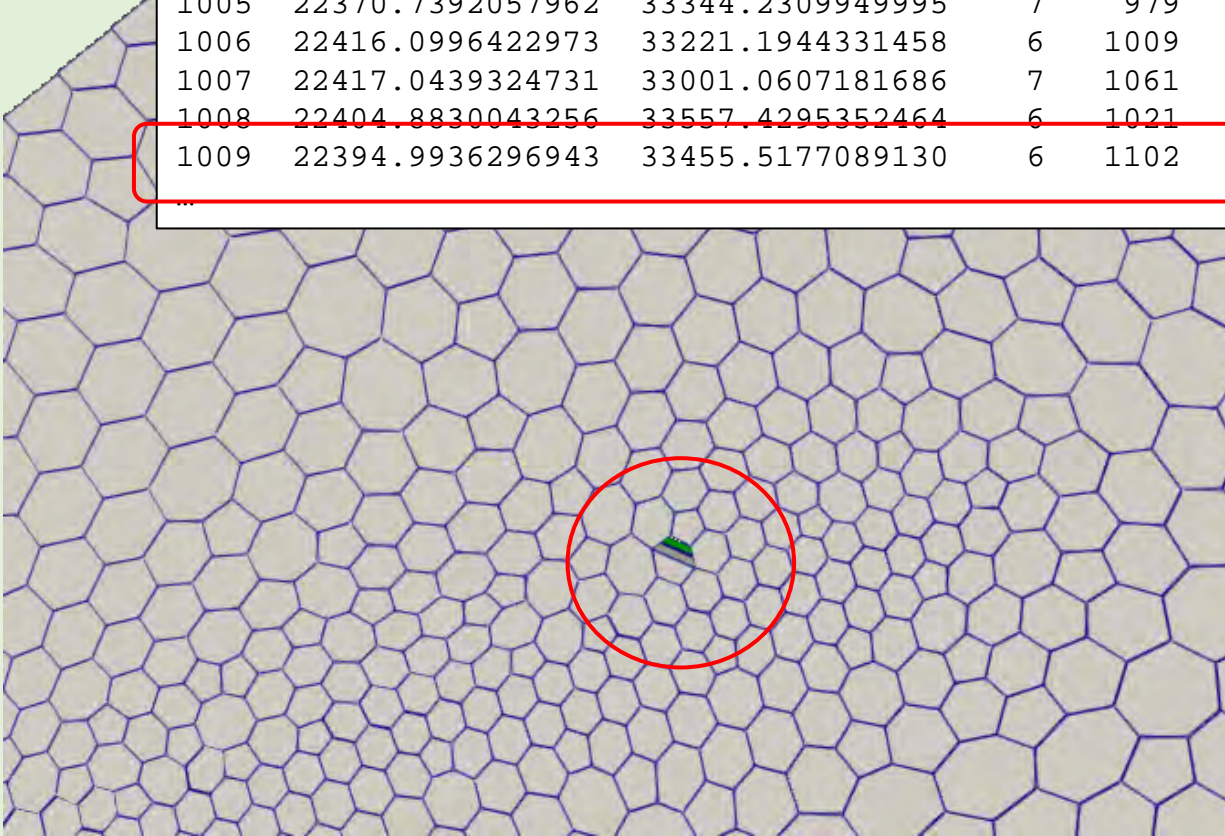
The file data checks out correctly — all face references, offsets and connectivity are consistent. Let me look at this from a different angle: check if there's anything geometrically unusual about cell 1009 (MF6 icell2d=1009, VTK cell 1008 if 0-indexed) and whether vertex 1102 — which it shares with cell 1008 — could hint at an overlap or mesh gap:

In other words. "Its not my fault; fix it yourself".

Working with Voronoi grids

part of the DISV file

```
...
1002 22314.7072592760 33247.5866627590 5 955 1039 1009 996 904
1003 22310.2210179656 33614.5021727621 6 899 944 1050 1021 983 972
1004 22320.7554441333 33521.3448110828 5 972 983 985 1012 919
1005 22370.7392057962 33344.2309949995 7 979 1011 1148 1081 1147 1039 955
1006 22416.0996422973 33221.1944331458 6 1009 1039 1147 1073 1146 1080
1007 22417.0439324731 33001.0607181686 7 1061 1145 1092 1090 1144 1097 990
1008 22404.8830043256 33557.4295352464 6 1021 1151 1150 1102 985 983
1009 22394.9936296943 33455.5177089130 6 1102 1149 1148 1011 1012 985
```



So I checked the DISV file for vertex/cell relationships, and verified that these relationships were ok using files that VOROGRIDGEN writes, imported into SURFER/GIS.

Working with Voronoi grids

I asked Claude

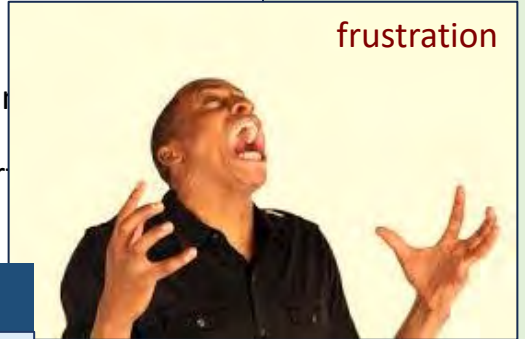
I'm sorry but I do not understand. In the DISV file *model.disv*, cell 1009 (the last cell in the CELL2D block) is defined by 6 vertices. This forms a valid polygon doesn't it?

Claude thought and thought...

You're right to question this — let me go back and read the actual last few lines of the CELL2D block directly from the file rather than relying on my parsing.

You're absolutely right — let me re-read those last lines directly and re-run my parser carefully to see exactly what it's getting

Found it. The file has cell 1009's data split across two lines — the parser reads one line at a time and gets cut off after 4 vertices. Let me check the exact byte content around that line:



Lessons learned

- Claude can make mistakes.
- Do not hesitate to question it
- But you have to know what you are doing (including knowing MODFLOW 6 conventions and file protocols)

[1012, 985].

The file is truncated — two vertex IDs and END CELL2D are missing. Let me fix *model.disv* and regenerate the VTK:

The Read tool (Windows native) shows the complete file with all 6 vertices. The Linux mount must have been showing a stale cached view. Let me verify the mount is in sync and regenerate the VTK:

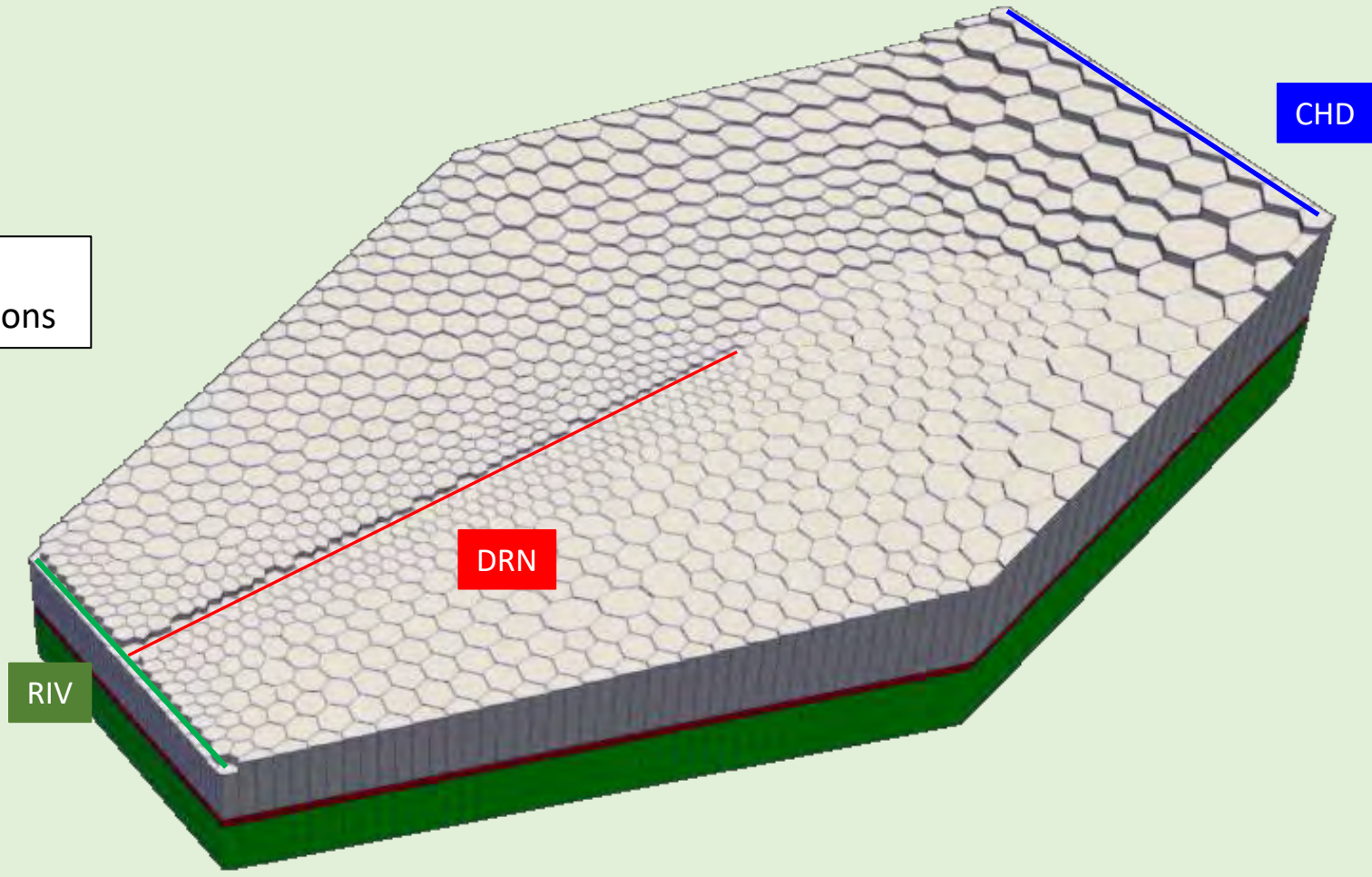
The Linux mount is lagging behind — the Windows file has 6 vertices + END CELL2D but the mount shows the old truncated version. I'll generate the VTK with the correct vertex list for cell 1009 hardcoded (since we've confirmed it's a valid polygon).
matter:

In other words, it's Claude's fault, but Claude blames WINDOWS.

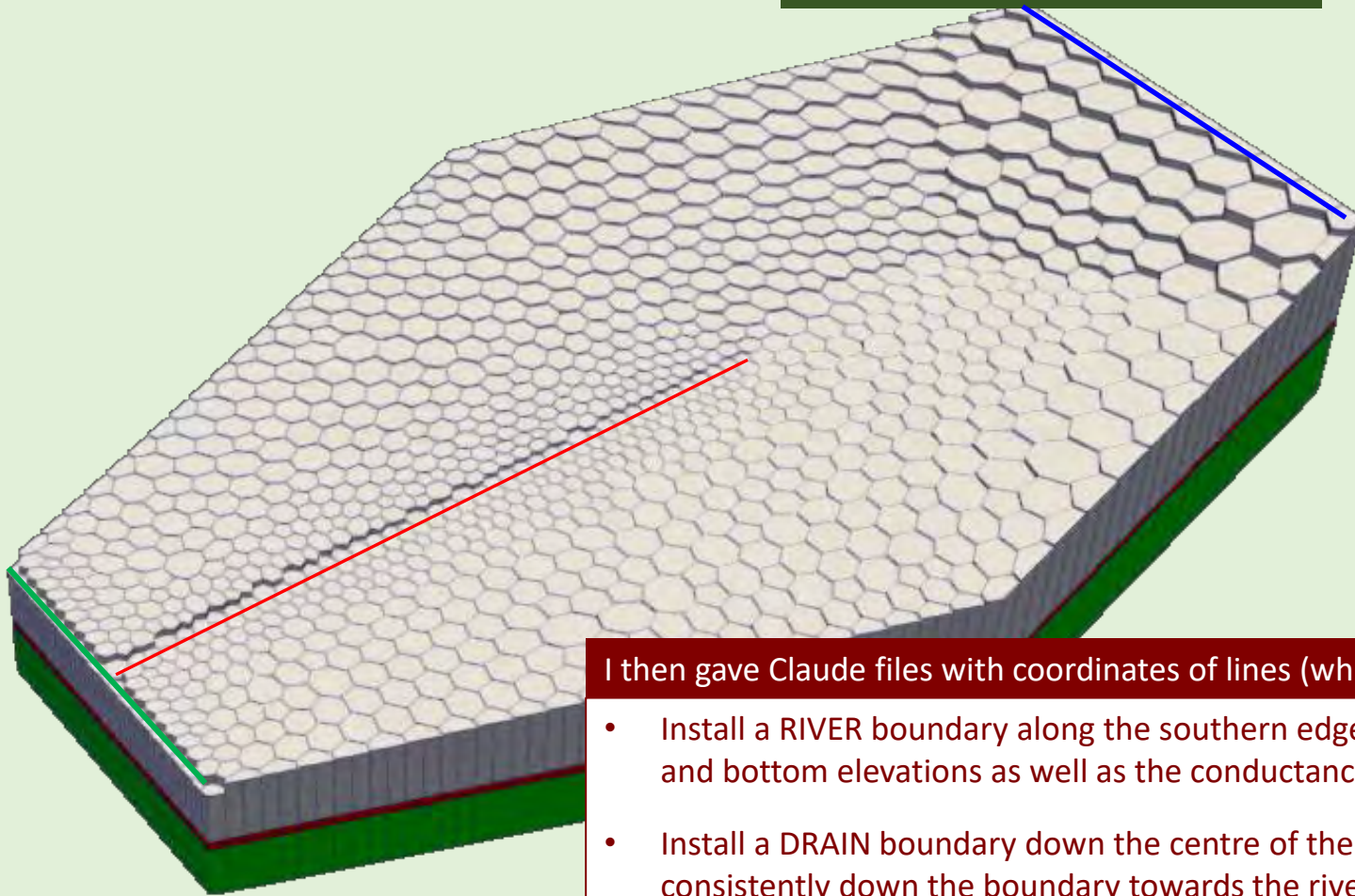
Working with Voronoi grids

This is nice; layering looks good

Cells where I would like to
emplace boundary conditions



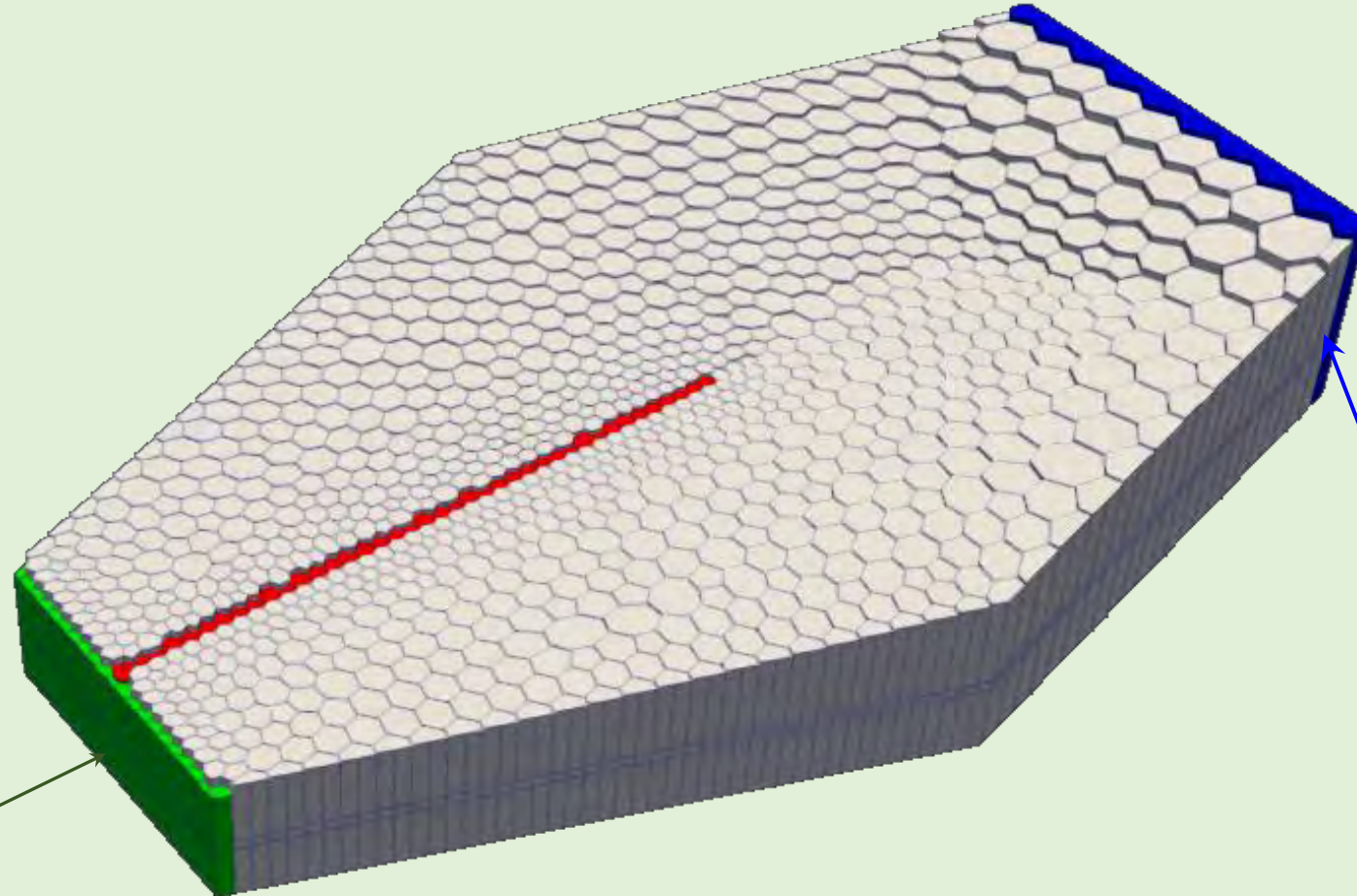
Working with Voronoi grids



I then gave Claude files with coordinates of lines (which I digitized in a GIS), and asked it to:

- Install a RIVER boundary along the southern edge of the model domain in layer 1 only. (I provided river head and bottom elevations as well as the conductance of all RIVER cells.)
- Install a DRAIN boundary down the centre of the model domain. Make sure that elevations decrease consistently down the boundary towards the river. Assign elevations to drains in accordance with this monotonic elevation decrease. Adjust the model top elevation referenced in the DISV file to accord with DRAIN elevations determined in this way.
- Install a CHD boundary along the northern end of the model domain in layer 1 only. I provided the uniform head to ascribe to this boundary.
- Write/alter all pertinent MODFLOW 6 input files.
- Produce a VTK file in which these boundaries are coloured different colours for easy identification.

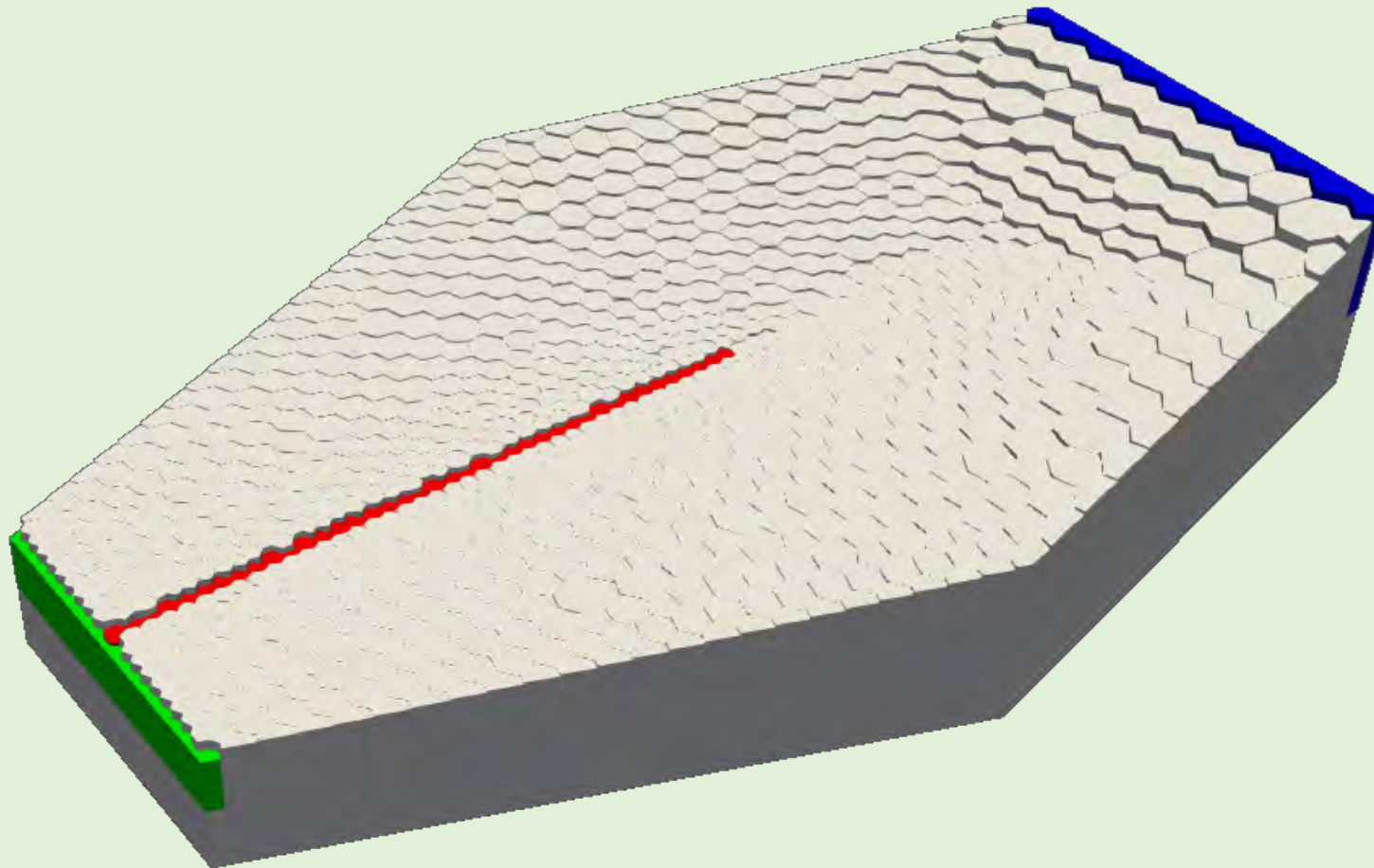
Working with Voronoi grids



Wrong!! Claude wrote the MODFLOW 6 input files correctly, but the picture says that boundaries extend through all model layers.

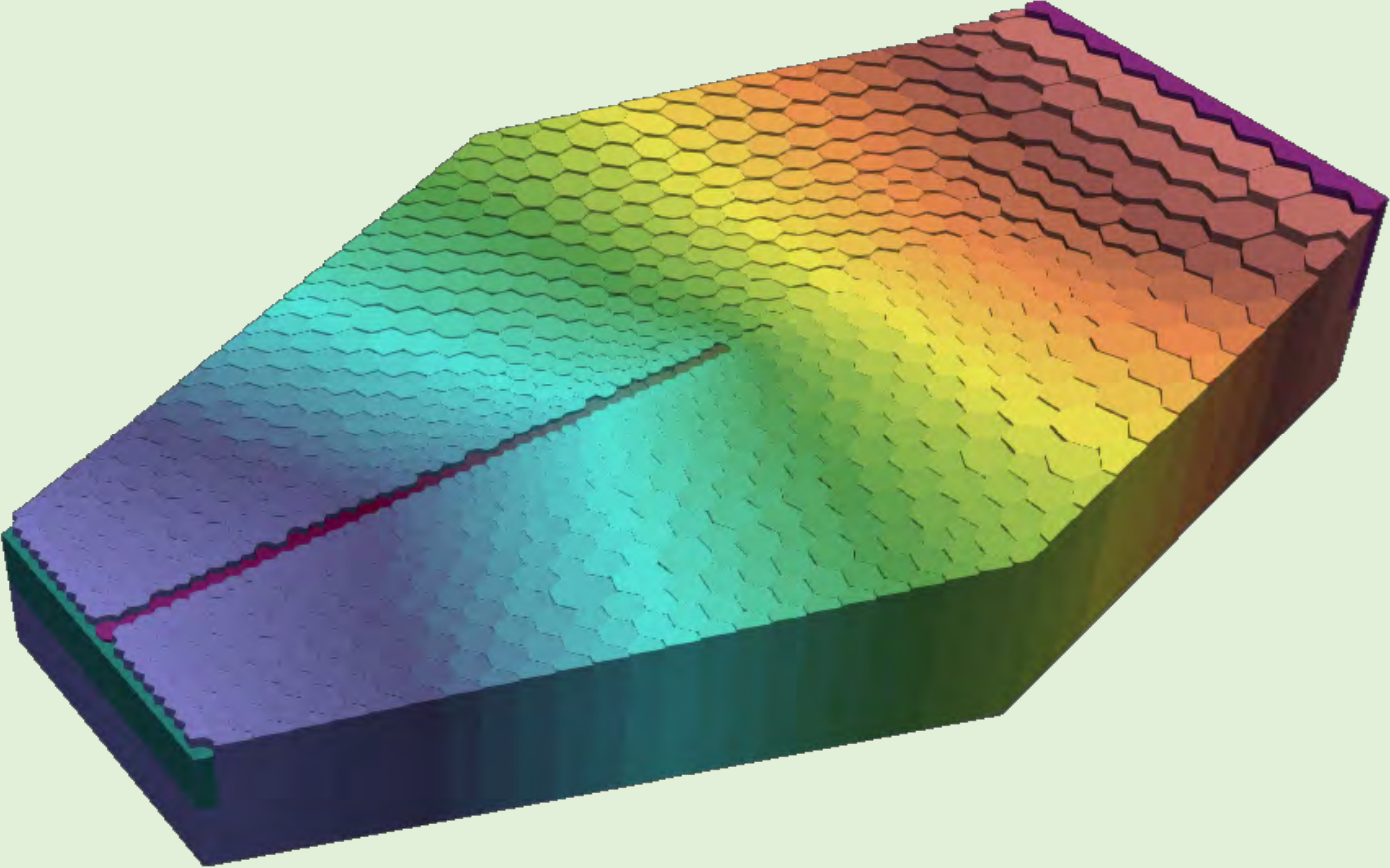
I explained this to Claude and asked it to try again.

Working with Voronoi grids



Then I ran MODFLOW 6 and asked Claude to provide a VTK file that allowed me to visualize MODFLOW-calculated heads.

Working with Voronoi grids



Writing Utility Programs

PEST

Model Independent Parameter Estimation & Uncertainty Analysis

PEST: Model Independent Parameter Estimation & Uncertainty Analysis

"PEST" refers to a set of essential tools in d

PEST

Model Independent Parameter Estimation & Uncertainty Analysis

Programs

All executable programs downloaded from this page are compiled to Linux versions. We can send you source code and makefiles for the LUMPREM and some of the programs of the Groundwater Utilities su

Groundwater Utilities

[Download the PEST Groundwater Utilities](#)

What's in the zip file?

- All programs comprising the PEST Groundwater Utility suite;
- A three-part manual for the PEST Groundwater Utilities.

PLPROC

[Download PLPROC](#)

What's in the zip file?

- A 64 bit version of PLPROC;
- Documentation of PLPROC.

OLPROC

[Download a OLPROC](#)

What's in the zip file?

- A 64 bit version of OLPROC;
- Documentation of OLPROC.

TS6PROC

[Download TS6PROC](#)

Manuals for PEST Groundwater Utilities

Groundwater Data Utilities

Part A: Overview

Watermark Numerical Computing

August, 2021

Groundwater Data Utilities

Part B: Program Descriptions

(See Part C for a description of programs pertaining to unstructured grid models.)

Watermark Numerical Computing

March, 2025

Groundwater Data Utilities

Part C: Programs Written for Unstructured Grid
Models

Watermark Numerical Computing

November, 2025

PEST

Parameter estimation & uncertainty analysis

Decision support modelling



- Overview of these pages
 - PEST and PEST++
 - The PEST suite
- Decision-support modelling
 - Environmental simulation
 - Metrics
 - Uncertainty
 - Understanding the problem
 - Problem decomposition
- Model calibration
 - Regularisation
 - Manual regularisation
 - Highly parameterised inversion
 - Subspace methods
 - Tikhonov regularisation
 - Why calibrate?
 - Dancing with models
 - PEST-model communication
 - Model run parallelisation
- PEST control file
- Parameters
- Observations
- Linear analysis
- Working with MODFLOW
- Pilot points
- Random parameters
- Nonlinear uncertainty analysis

Navigation: [Overview of these pages](#) >

The PEST suite

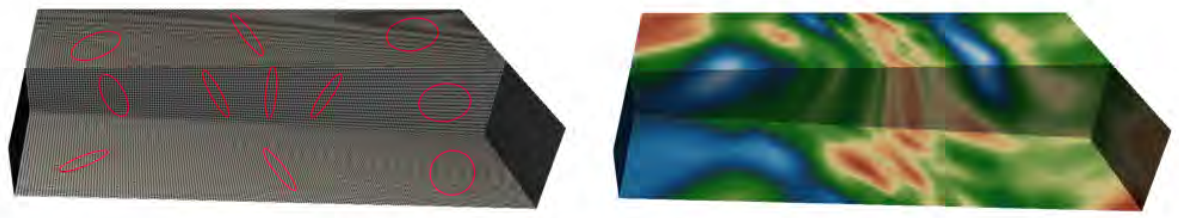
measurements were made. It can formulate secondary model outputs/observations from primary model outputs/observations. It writes output files that PEST can read, and instructions that teach PEST how to read them. It calculates weights to assign to different observations and observation types. Equations of arbitrary complexity can be used in weights formulation.

Groundwater utilities

The Groundwater Utility suite is comprised of over 170 programs. Some of these have been superseded by programs such as PLPROC and OLPROC. Others provide functionality that can be found nowhere else. The groundwater utility suite is under constant development.

Roles performed by different members of this suite include the following:

- zonal and pilot point parameterisation
- construction of 2D and 3D spatial covariance matrices used for pilot point regularisation and uncertainty analysis based on stationary and nonstationary variograms
- construction of cell-based 2D/3D stochastic fields based on stationary/nonstationary variograms for structured/unstructured grids
- spatial and temporal interpolation from structured/unstructured grids to sites at which measurements were made
- reading of MODFLOW, MODFLOW 6 and MODFLOW-USG binary dependent variable and budget files
- extraction of arrays from MODFLOW, MODFLOW 6 and MODFLOW-USG input files
- provision of a rudimentary interface between MODFLOW, MODFLOW6 and MODFLOW-USG models on the one hand and display/processing platforms such as SURFER, GRAPHER, PARAVIEW and GIS packages on the other hand
- automation of PEST input dataset construction for complex measurement datasets



A 3D stochastic hydraulic property field governed by a spatially varying covariance function.

Utilities for structured grids

ALPHABETICAL LISTING OF GF (See Part C of the manual for the Gro documentation of unst

adiobs	Adjusts observation weight control file according to us
anis2vtk	Creates a VTK file through many points. A VTK file visualisation.
arr2bore	Undertakes spatial interpol
atravobs	Facilitates the introduc MODFLOW/MT3D-comp estimation process.
atrdt	Lists the contents of head/drawdown/concentrat
bud2hyd	Extracts flow data from term file. Rewrites this dat
bud2smp	Extracts flow data from term file. Rewrites this dat
bud2smp1	Similar to EUD2SMP, b number of layers.
conc2elev	Computes the elevation of of a sequence of concentra
dar2smp	Translates system states c file format.
dbl2sgl	Re-writes a binary, head/drawdown file as a si
elev2conc	Computes a sequence of it layer) based on a user-sup array, and (spatially varyin
elev2concl	Similar to elev2conc, but c
fac2fefl	Uses PPKFAC_FEFL-gene model input data file on t pilot points.
fac2fem	Uses ppk2fac-generated k file on the basis of spatial i
fac2g	Complements PPK2FAC recording interpolated valu

fac2real	Uses PPKFAC-generated compatible real array on pilot points.
fac2real3d	Uses PPK2FAC3D-gene MODFLOW-compatible r set of three-dimensional p
fac2rsm	Uses PPKFACR-generate input data file on the ba points.
fem2smp	Converts MicroEEM outp
fieldgen	Generates a stochastic fie sequential Gaussian simul
fieldgen_sva	Generates stochastic field Hyper-parameters that g MODFLOW-compatible (varying anisotropy.)
fieldgen2d_sva	Generates a 2D stochastic model. Hyper-parameters the model domain; station
fieldgen2d_sval	Similar to FIELDGEN2D integration takes place do stochastic fields.
fieldgen3d_sva	Generates a 3D stochast unstructured grid model. vary throughout the mode
fieldgen3d_sval	Similar to FIELDGEN3D integration takes place do stochastic fields.
fieldgen3d_sva2	Similar to FIELDGEN3D over a user-supplied set parameter estimation.
genreal2srf	Interpolates from a MOD nodes of a SURFER grid f
genreg	Inserts prior informatio regularisation into an exist
getmularr	Extracts arrays from MO user-nominated simulatio formatted files.

getmularr1	Extracts all arrays for a non MODFLOW/MT3D unformatted ou unformatted MODFLOW/MT3D out
grid2arc	Writes ARCINFO generate files of t grid as defined by a user-supplied int
grid2bln	Writes a SURFER blanking file of th grid as defined by a user-supplied int
grid2dxf	Writes a DXF file of the active p defined by a user-supplied integer arr
grid2pt	Tabulates the coordinates of the cell within an active window defined by s
hans2bln	Creates a SURFER BLN file showing
iidsen	Generates random numbers FIELDGEN2D_SVA1 and FIEL
int2mif	Generates MAPINFO MIF as MODFLOW/MT3D-compatible inte
int2real	Builds a MODFLOW/MT3D-compa of a MODFLOW/MT3D-compatible
lay2diff	Evaluates head value differences in a bore sample file, bore coordinates f
logarray	Evaluates the log (to base 10) of all e
many2one	Splits MODFLOW/MT3D-generate multiple two-dimensional res formatted/unformatted files.
mf2vtk	Writes a "legacy VTK" file through v and calculated system states can PARAVIEW.
mf2vtk1	Similar to MF2VTK. However it rea or a number of tabular data files and cells.
mkppstat	Writes a pilot point statistical s PPCOV_SVA in which local vario point density.
mkppstat3d	Writes a pilot point statistical s PPCOV3D_SVA in which local v "a_xxx" values reflect local pilot poi

mod2array	Reads a MODFL arrays from that fil
mod2obs	Interpolates mode those cited in a sample file.
mod2obs_dbl	Same as MOD2O system state outpu model).
mod2smp	Interpolates the MODFLOW/MT3 the bore-specific d
mod2smpdiff	Interpolates the MODFLOW/MT3 difference or ratio of bores.
PARCOV	Builds a geostatist whose coordinates
parm3d	Assists in the pilo hydrogeologic uni
pestprep	Automates constru for a model com MOD2OBS, or 1 SMP2SMP.
pestprep1	Similar to PESTP naming.
pestprep2	Similar to PESTP added to an existin
pmp2info	Builds a bore inf containing cumula for a user-supplied
pmpchk	Checks the integri
ppcov	Builds a covarian one or a number o
ppcov3d	Builds a covarian parameters based o
PPCOV_SVA	Builds a covarian parameters under properties, can var

ppcov3d_sva	Builds a covariance matrix pertaining to three-dimensional pilot point parameters under the assumption that anisotropy, and other geostatistical properties, can var
ppk2fac	Calculates kriging pilot points to mod
ppk2facf	Calculates kriging pilot points to the
ppk2facg	Calculates kriging where the coordin three vertical colu
ppk2fac1	Identical to ppk2f which it writes is s
ppk2fac2	Identical to ppk2f radius.
ppk2fac3	Contains improve thin alluvial mode exist.
ppk2fac3d	Calculates krigin dimensional pilot arrays.
ppk2facr	Calculates kriging pilot points to th protocol is identic
ppk2fac_fefl	Calculates kriging pilot points to the protocol is identic
ppkreg	Adds a "prior in control file where
ppkreg1	Similar to ppkreg "smoothness regu regularisation".
ppsamp	Used in calibration fields at pilot poi generates differen
pt2array	Builds a MODFL array element is c within the respect

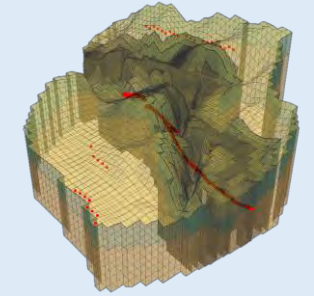
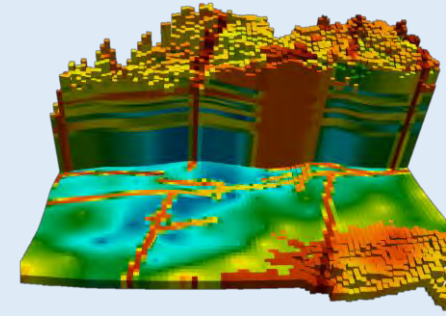
ptingrid	Locates the finite-difference cells in which arbitrary, user-supplied points lie; optionally provides the value of an integer or real array element pertaining to the cell containing each such point.
rdat2tab	Reads an RSM element data file or index file. Adds mesh centroid coordinates to respective data elements and re-writes data in tabular format.
real2int	
real2mif	
real2srf	
real2tab	
rearray	
rotbln	
rotdat	
rotdef	
rsm2srf	
section	
smp2hyd	
smp2info	

smp2smp	Interpolates data contained within one bore sample file to the dates and times represented in another bore sample file.
smpcal	Calibrates one time-series dataset on the basis of another.
smpchk	Checks the integrity of a bore sample file.
smpdiff	Writes a new bore sample file in which differences are taken between successive values in an existing bore sample file, or between values in an existing file and a reference value.
smpstat	Reads a bore sample file. Writes a text file containing some statistics for data associated with each bore.
smptrend	Writes a new bore sample file in which differences are taken between samples within an existing bore sample file and either the first sample for each bore in that file or a reference sample. However sampling is restricted to a yearly sample window.
srf2real	Re-writes a SURFER grid file as a MODFLOW-compatible real array.
tab2int	Generates a MODFLOW/MT3D-compatible integer array from an integer array stored within a GIS.
tab2real	Generates a MODFLOW/MT3D-compatible real array from a real array stored within a GIS.
tabconv	Translates between integer or real array table files using row/column identifier format and those using cell number identifier format.
tabstat	Combines two real arrays by addition, subtraction, multiplication, division and partial replacement.
vertreg	Adds "vertical regularisation" prior information to a PEST control file where parameterisation is based on pilot points.
zone2bln	Writes a SURFER "blanking" file of finite-difference grid zonation as defined by a user-supplied, MODFLOW-compatible integer array.
zone2dxf	Writes a DXF file of finite-difference grid zonation as defined by a user-supplied, MODFLOW-compatible integer array.

PEST Groundwater Utilities



the “model sandwich”



Many utility programs contribute to PEST/model interface

- spatial/temporal interpolation to sites/times of borehole measurements
- spatial and temporal differencing
- handling of pilot points
- stochastic field generation (stationary/nonstationary statistics)
- building a PEST input dataset
- etc

Others perform tasks that are useful for modelling in general

- display contents of MODFLOW binary output files
- accumulation of budgets
- importing parts of model input datasets from data hosts (e.g. GIS)
- exporting model properties/results to a GIS
- exporting model properties/results to 3D viewing platform such as PARAVIEW
- etc

Good things

- Written in FORTRAN so are fast, transportable and do not rely on version-compatible modules/libraries.
- Undertake a plethora of tasks that are not supported by some/all graphical user interfaces or FloPy/PyEMU.
- Good at error-checking and well documented.

Bad things

- Each utility only performs part of a typical workflow.
- A user finds it hard to know what utility program to use for a specific task.
- Often the utilities don't “fit together” well (i.e. they are too “atomic”) so that common tasks require manual intervention. (Users can make mistakes).
- There are some tasks that are difficult to do using FORTRAN programs (e.g. writing of shapefiles and binary VTK files).

Using Claude to replace and enhance utilities

I asked Claude

Please write a program (in **FORTRAN**) that does the following:

- Reads a binary grid file (GRB file) written by MODFLOW 6
- Works out:
 - whether the grid is DIS or DISV
 - number/disposition of cells and layering
 - the coordinates of all vertices of all active cells
- Reads a binary dependent-variable (e.g. heads) file written by MODFLOW 6
- Writes a shapefile recording heads in all layers of the model at a user-specified simulation time.

This could be Python, C, C++ or your language of choice.

Running Claude's program...

```
E:>.\programs\make_shape
Binary grid file (GRB): ex-gwf-sagehen.dis.grb
Dependent variable (binary head) file: ex-gwf-sagehen.hds
TOTIM for which heads are required: 40.0
PROJ file (press <Enter> for none): <Enter>
Output filename base: test

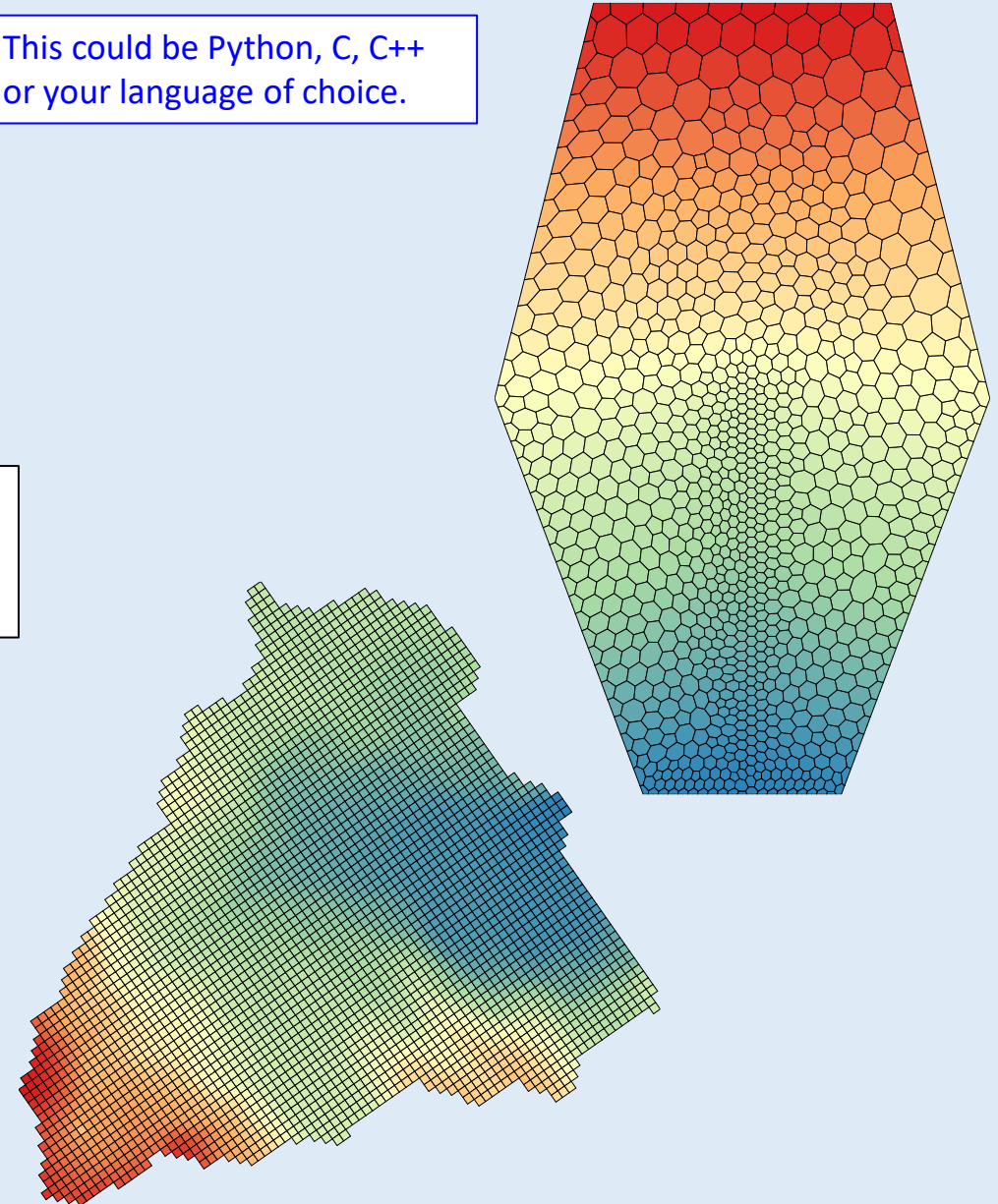
Reading grid from : ex-gwf-sagehen.dis.grb
Grid type       : DIS
Layers (NLAY)   : 1
Rows x Cols     : 73 x 81
Origin          : ( 45235.0, 763422.0) ANGROT= 35.0 deg

Reading heads from: ex-gwf-sagehen.hds

Writing shapefile : test.shp/.shx/.dbf ...
Cells to write    : 3387 (active in >=1 layer)
No PROJ file given -- .prj not written (coordinates undefined).
Done.
```

Not pretty but:

- effective
- informative
- forgiving of errors

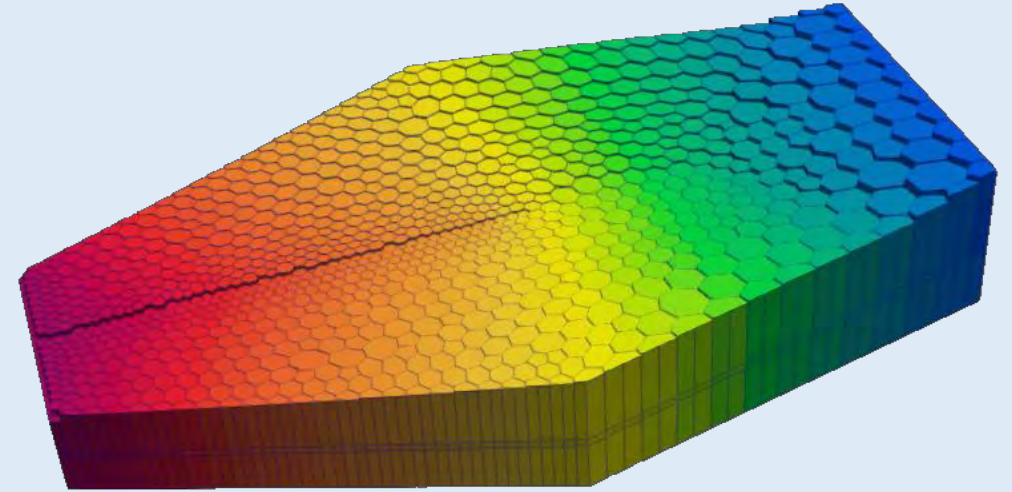


Using Claude to replace and enhance utilities

Then I asked Claude

Please write a program (in FORTRAN) that does the following:

- Reads a binary grid file (GRB file) written by MODFLOW 6
- Works out:
 - whether the grid is DIS or DISV
 - number/disposition of cells and layering
 - the coordinates of all vertices of all active cells
- Reads a binary dependent-variable file written by MODFLOW 6
- Writes a VTK file so that I can display/colour in PARAVIEW:
 - the grid
 - heads
 - layer number



Running Claude's program...

```
E:>.\programs\make_vtk
Binary grid file (GRB): model.disv.grb
Dependent variable (binary head) file: model.hds
TOTIM for which dependent variable is required: 1.0
Output filename base (writes <base>.vtu): test

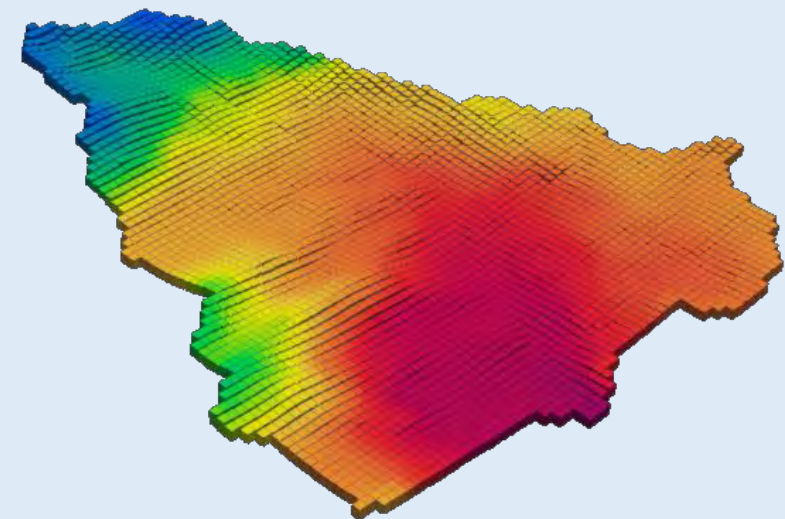
Reading grid from : model.disv.grb
Grid type       : DISV
Layers (NLAY)   : 3
Cells/layer, verts: 1009, 2118

Reading heads from: model.hds

Writing VTK file : test.vtu ...
Solid cells to write: 3027 (one per active layer-cell)
Done.
```

Not pretty but:

- effective
- informative
- forgiving of errors



Using Claude to replace and enhance utilities

Then I asked Claude

Please write a manual for the programs that you have just written (in Microsoft Word).

make_shape.f90 and make_vtk.f90
MODFLOW 6 post-processing utilities — program docs

1. Overview

Both programs are interactive command-line FORTRAN utilities that post-process a MODFLOW 6 simulation and export the results to a GIS-ready shapefile and a VTK mesh, respectively. Both read the same two MODFLOW 6 input files: a binary grid file (`grb`) and a binary dependent-variable file (typically a head file, `hds`).

2. Shared design

The two programs do not share a compiled module or library file, consistent with the other utilities in programs/. They do, however, share several subroutines (copied between the two files), so that future program updates can reuse the same logic rather than re-deriving it.

3. make_shape.f90

3.1 Purpose

Exports dependent-variable values (typically heads) at a single simulation time to a GIS-ready shapefile.

3.2 Running the program

The program takes no command-line arguments; it prompts for everything it needs interactively:

- Binary grid file (`grb`) — path to the model's `grb` file.
- Dependent variable (binary head) file — path to the `hds` (or similar) file for which heads are required — a simulation time. If it does not match the file exactly, the program warns and uses the nearest available file.
- PROJ file — path to a text file containing a coordinate system definition to be copied into the output. Press Enter for none, in which case the shapefile's coordinate system is left undefined.
- Output filename base — the shapefile is written as `<base>_shp`, `<base>_tbl`, and `<base>_prj` if a PROJ file was supplied.

3.3 Output

One polygon is written per plan-view model cell that is active (DOMAIN = 1) in the requested time slice. Each polygon carries the following DBF attribute fields:

Field	Grid type	Meaning
ROW, COL	DIS only	1-based row and column indices
ICELL2D	DISV only	1-based cell index
HEAD_LO1, HEAD_LO2, ...	Both	Dependent variable value for the requested time slice

3.4 Compiling

4. make_vtk.f90

4.1 Purpose

Exports dependent-variable values at a single simulation time to a solid 3-D VTK mesh (not just a flat plan view).

4.2 Running the program

Also fully interactive, prompting in this order:

- Binary grid file (`grb`)
- Dependent variable (binary head) file
- TOTIM for which dependent variable is required
- Output filename base — writes `<base>_vtk` files

(There is no PROJ-file prompt for this program; `grbview` files do not carry a definition in the way a shapefile does.)

4.3 Output: a solid VTK mesh

Every active model cell (IDOMAIN > 0), in every layer, is written as one solid VTK cell (type 42) — a vertical prism standing on the cell's plan-view footprint, with its top at the layer's TOP elevation (TOP of layer 1, or the layer's elevation). Both elevations are read directly from the `grb` file. Because the VTK file is written in the VTK XML format, it shows the actual 3-D model grid thickness, stacked correctly — rather than a stack of flat sheets.

The file format is VTK XML (preferred), not the older legacy ASCII format. The latter is a legacy-format reader has no general polyhedron support with an arbitrary number of side faces (4 for a DIS rectangle, however many faces a DISV cell has). The face-structure algorithm only represents the newer XML format. The face-structure algorithm between cells, a bottom polygon, a reversed top polygon, and one quadrilateral face. The face-structure algorithm is already validated for this project's `vsctc/model_cells.vtu` and `vsctc/model_cells.vtu`.

Two scalar fields are attached to every cell as `CellData`, so cells can be colored by layer and by value.

Field	Type	Meaning
LAYER	Integer	Model layer number (1 = top layer)
HEAD	Double	Dependent variable value for the requested time slice (-999.0 if missing / no-flow / dewatered)

4.4 Compiling

5. Binary file formats relied on

For reference, both programs depend on the following MODFLOW 6 binary format documents and by reading real `grb`/`hds` files byte-for-byte.

5.1 Binary grid file (`grb`)

- Four fixed 50-byte text header lines: "GRID DIS" (or "GRID DISV"), "VERTICES", "LENTXT n".
- n variable-definition lines, each a fixed LENTXT bytes (100 in practice), of the form `ndim1 dim1 dim2 ...`.
- Then the binary data for each variable, in the same order as the definition lines.
- DIS grids declare 16 variables (NCELLS, NLAY, NROW, NCOL, NIA, XORIGIN, DELR, DELC, TOP, BOTM, IA, JA, IDOMAIN, ICELLTYPE).
- DISV grids declare 20 (as above, but NCELL/NVERT/NIAVERT replace NROW, NVERTICES, CELLY, IAVERT, JAVERT).

5.2 Binary dependent-variable file (`hds`)

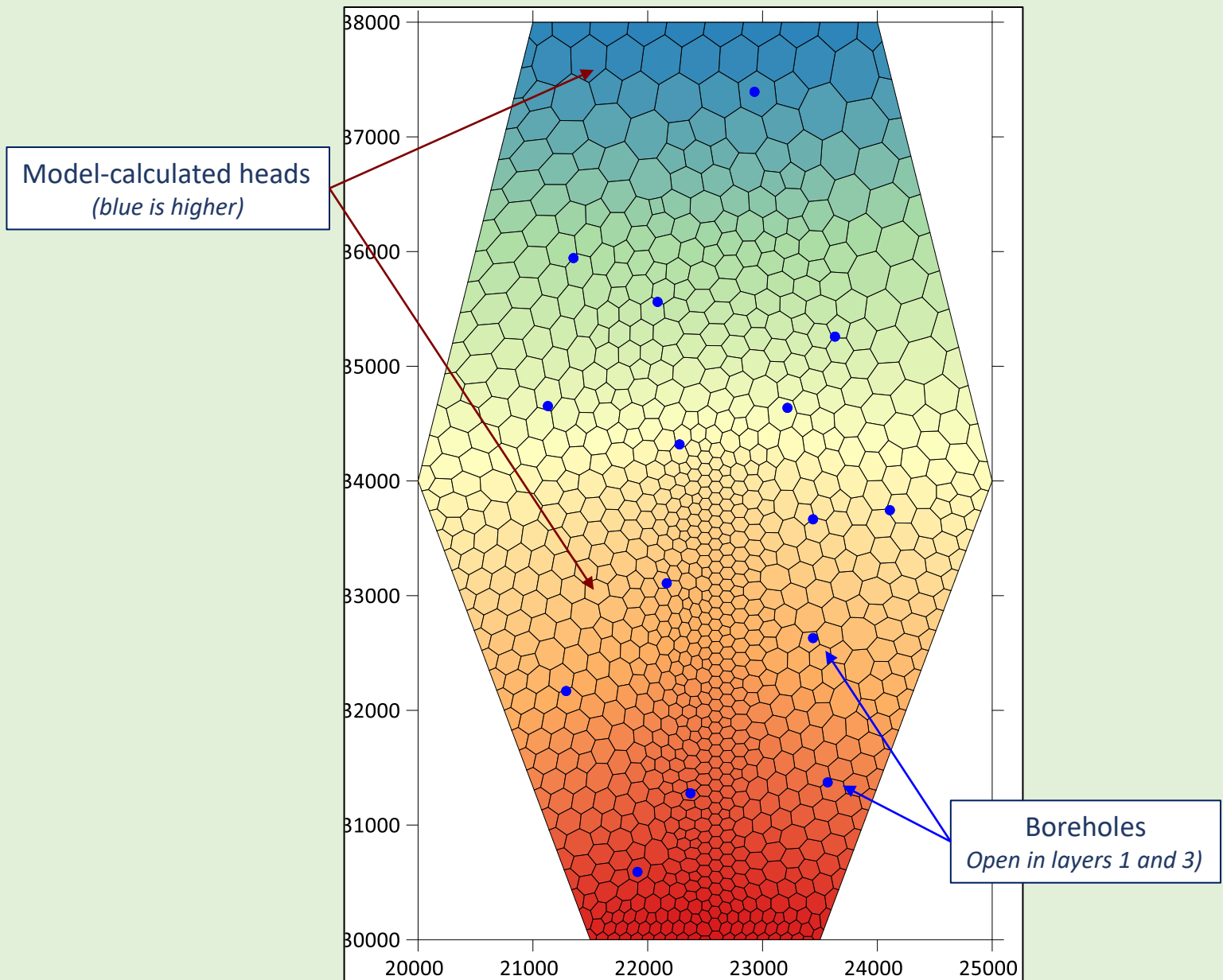
Identical per-layer record structure for both grid types: a 52-byte header (KSTP, 16-character TEXT label, then NCOL, NROW, ILAY) followed by NCOL*NROW double values (one record is written per layer).

6. Limitations and testing notes

- No Fortran compiler was available in the `gwork` sandbox these programs were compiled there. Both were reviewed closely by hand and cross-checked their algorithms to Python and running them against two real test models (`vsctc/mf6` (a DIS grid, rotated 35°, with inactive cells) and `vsctc/mf6` (a DISV grid, with a polyhedron structure) the face-array bookkeeping. Compiling programs in a real Fortran environment, and opening the results in QGIS, is recommended final check.
- A cell footprint is only written if the cell is active in at least one layer (more specifically: `make_vtk.f90`; fully inactive columns are omitted entirely).

Spatial Interpolation from a Voronoi Grid

Interpolating from a Voronoi Grid



File bores.crd			
b1u	21353.183	35943.216	1
b2u	22085.871	35560.944	1
b3u	23630.888	35258.312	1
b4u	24108.728	33745.151	1
b5u	23439.752	32630.191	1
b6u	23567.176	31371.879	1
b7u	21910.663	30591.406	1
b8u	22372.575	31276.311	1
b9u	21289.471	32168.279	1
b10u	22165.511	33108.031	1
b11u	23439.752	33665.511	1
b12u	21130.191	34653.048	1
b13u	22930.056	37392.665	1
b14u	23216.760	34637.120	1
b15u	22277.007	34318.560	1
b11	21353.183	35943.216	3
b21	22085.871	35560.944	3
b31	23630.888	35258.312	3
b41	24108.728	33745.151	3
b51	23439.752	32630.191	3
b61	23567.176	31371.879	3
b71	21910.663	30591.406	3
b81	22372.575	31276.311	3
b91	21289.471	32168.279	3
b101	22165.511	33108.031	3
b111	23439.752	33665.511	3
b121	21130.191	34653.048	3
b131	22930.056	37392.665	3
b141	23216.760	34637.120	3
b151	22277.007	34318.560	3

Interpolating from a Voronoi Grid

I asked Claude

I want to undertake spatial interpolation of heads from a Voronoi grid to the sites of boreholes that are listed in file *bores.crd*. Are there any Groundwater Utility Suite programs that do this? Or would you prefer to do this yourself?

Claude said

On the task — yes, there's a purpose-built pathway, and it's a two-step one for a DISV grid:

1. **MF6INTFAC** reads `model.disv.grb` + `bores.crd`, works out which connected cell centres enclose each bore, and writes a node-to-bore interpolation factor file. For DISV it does proper barycentric interpolation from an enclosing triangle (or quad/polygon fallback) of cell centres — not just nearest-cell.
2. **MF6MOD2SMP** then reads those factors + `model.hds` and writes the interpolated heads as a bore sample file. (It needs a `settings.fig` in the directory.)

First I parse `model.disv.grb` (the binary grid file MODFLOW wrote). From it I get, per layer, the cell-centre coordinates (`CELLX / CELLY`), the cell vertices (`VERTICES , IAVERT / JAVERT`), and the cell-to-cell connectivity (`IA / JA`), plus `TOP / BOTM` .

Then, for each bore (name, X, Y, layer) in `bores.crd` :

1. Find the home cell — the DISV cell whose polygon actually contains (X, Y), by point-in-polygon test on that cell's vertices. That's the MF6PTINGRID step.
2. Assemble candidate nodes — the home cell centre plus its horizontally-connected neighbours' centres (from `IA / JA` , restricted to the bore's layer).
3. Interpolate, in the same priority order MF6INTFAC uses:
 - find a triangle of connected cell centres that encloses the point → barycentric weights;
 - if no triangle, a quadrilateral → average of its two enclosing triangles' barycentric results;
 - if neither (typically a near-boundary point), fall back to inverse-distance-squared from the surrounding centres.

The weights always sum to 1.

4. Read heads for that bore's layer from `model.hds` at the candidate nodes and apply the weights → the interpolated head at the bore.

I'll flag any bore that can't be enclosed (outside the grid or too near the edge) rather than silently guessing, and handle dry/inactive nodes by dropping them and re-normalising the weights (MF6MOD2SMP's "reapportion" option).

Output will be a bore sample file plus a CSV with head, home-cell index, and which interpolation type was used. Let me build it.

Then Claude did the following

- Wrote keyboard input files for MF6INTFAC and MF6MOD2SMP.
- Wrote a batch file that allowed me to run MF6INTFAC and MF6MOD2SMP using a single command.
- Wrote and ran a Python Script that records interpolated heads in a site sample file with the same format as used by MF6MOD2SMP (and other Groundwater Utility Suite programs).
- Asked me to run the batch file and then offered to compare the two site sample files.

So Claude carried out the interpolation in two different ways – my way and its way.

Interpolating from a Voronoi Grid

Batch file written by Claude

```
@echo off
rem -----
rem  Interpolate MODFLOW 6 DISV heads to the bores in bores.crd using the
rem  Groundwater Data Utilities. Run from the voro folder. Requires settings.fig
rem  (already present) and mf6intfac.exe / mf6mod2smp.exe on your PATH.
rem -----

rem Step 1: build node-to-bore interpolation factors from the grid + bore coords
mf6intfac < mf6intfac.in
if errorlevel 1 goto :err

rem Step 2: interpolate heads from model.hds to the bores -> bores_mf6.smp
mf6mod2smp < mf6mod2smp.in
if errorlevel 1 goto :err

echo.
echo Done. Compare bores_mf6.smp (gwutil) with bores_heads.smp (Python).
goto :eof

:err
echo.
echo A program returned an error - check the screen output above.
```

Programs of the Groundwater Utility Suite

Interpolating from a Voronoi Grid

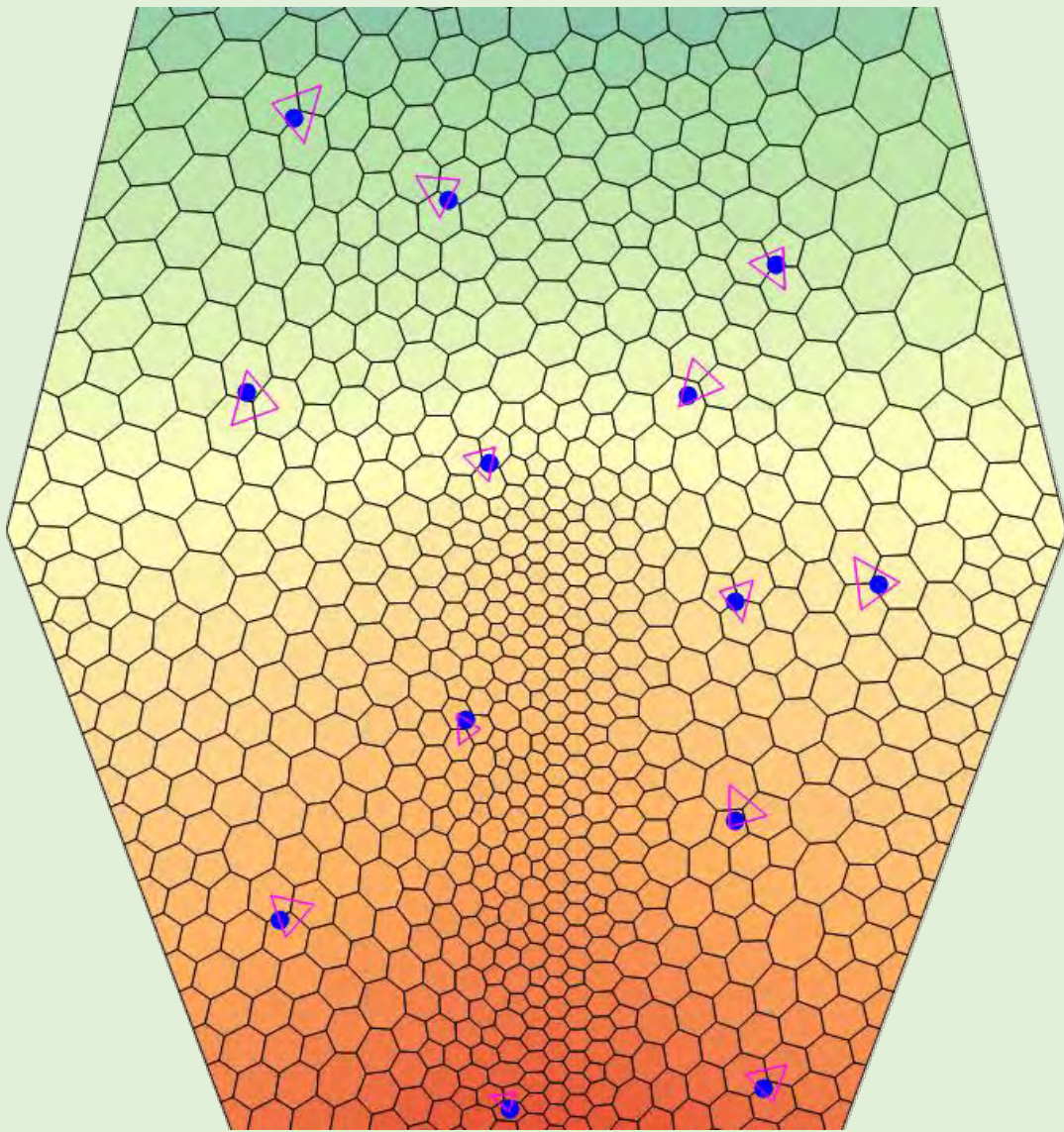
SSF file written by Claude's Python script

b1u	01/01/2000	00:00:00	1.135627e+02
b2u	01/01/2000	00:00:00	1.125239e+02
b3u	01/01/2000	00:00:00	1.115167e+02
b4u	01/01/2000	00:00:00	1.076936e+02
b5u	01/01/2000	00:00:00	1.044032e+02
b6u	01/01/2000	00:00:00	1.017219e+02
b7u	01/01/2000	00:00:00	9.949500e+01
b8u	01/01/2000	00:00:00	1.005604e+02
b9u	01/01/2000	00:00:00	1.036980e+02
b10u	01/01/2000	00:00:00	1.047663e+02
b11u	01/01/2000	00:00:00	1.069897e+02
b12u	01/01/2000	00:00:00	1.099303e+02
b13u	01/01/2000	00:00:00	1.183606e+02
b14u	01/01/2000	00:00:00	1.095328e+02
b15u	01/01/2000	00:00:00	1.083128e+02
b1l	01/01/2000	00:00:00	1.135646e+02
b2l	01/01/2000	00:00:00	1.125255e+02
b3l	01/01/2000	00:00:00	1.115174e+02
b4l	01/01/2000	00:00:00	1.076921e+02
b5l	01/01/2000	00:00:00	1.044042e+02
b6l	01/01/2000	00:00:00	1.017225e+02
b7l	01/01/2000	00:00:00	9.952019e+01
b8l	01/01/2000	00:00:00	1.006430e+02
b9l	01/01/2000	00:00:00	1.036972e+02
b10l	01/01/2000	00:00:00	1.048147e+02
b11l	01/01/2000	00:00:00	1.069921e+02
b12l	01/01/2000	00:00:00	1.099302e+02
b13l	01/01/2000	00:00:00	1.182962e+02
b14l	01/01/2000	00:00:00	1.095349e+02
b15l	01/01/2000	00:00:00	1.083501e+02

SSF file written by MF6INTFAC / MF6MOD2SMP

B1U	02/01/2000	00:00:00	113.5566706
B2U	02/01/2000	00:00:00	112.5239272
B3U	02/01/2000	00:00:00	111.5318771
B4U	02/01/2000	00:00:00	107.6952045
B5U	02/01/2000	00:00:00	104.3994269
B6U	02/01/2000	00:00:00	101.7234847
B7U	02/01/2000	00:00:00	99.49722077
B8U	02/01/2000	00:00:00	100.5582854
B9U	02/01/2000	00:00:00	103.6893655
B10U	02/01/2000	00:00:00	104.7685992
B11U	02/01/2000	00:00:00	106.9914829
B12U	02/01/2000	00:00:00	109.9397746
B13U	02/01/2000	00:00:00	118.3592726
B14U	02/01/2000	00:00:00	109.5340044
B15U	02/01/2000	00:00:00	108.2928764
B1L	02/01/2000	00:00:00	113.5587457
B2L	02/01/2000	00:00:00	112.5254850
B3L	02/01/2000	00:00:00	111.5325974
B4L	02/01/2000	00:00:00	107.6936000
B5L	02/01/2000	00:00:00	104.4003636
B6L	02/01/2000	00:00:00	101.7239242
B7L	02/01/2000	00:00:00	99.52057788
B8L	02/01/2000	00:00:00	100.6410914
B9L	02/01/2000	00:00:00	103.6884127
B10L	02/01/2000	00:00:00	104.8154986
B11L	02/01/2000	00:00:00	106.9933438
B12L	02/01/2000	00:00:00	109.9396791
B13L	02/01/2000	00:00:00	118.2955110
B14L	02/01/2000	00:00:00	109.5358806
B15L	02/01/2000	00:00:00	108.3276633

Interpolating from a Voronoi Grid



How MF6INTFAC interpolates from the Voronoi grid
(It writes a BLN file that allows display of triangles/quadrangles/polygons joining grid cell centres from which interpolation takes place.)

Interpolating from a Voronoi Grid

Claude's utility program provides a CSV file to explain the interpolation process

bore	x	y	layer	home_icp	interp_type	n_nodes	head	note	nodes_icpl	weights	min_contrib	max_contrib	home_head
b1u	21353.18	35943.22	1	487	triangle	3	113.5627	ok	487;339;484	0.7246;0.1	113.2430886	114.5730439	113.2430886
b2u	22085.87	35560.94	1	853	triangle	3	112.5239	ok	853;636;852	0.5300;0.0	112.2556976	113.1920869	112.2556976
b3u	23630.89	35258.31	1	471	triangle	3	111.5167	ok	471;322;582	0.1249;0.1	111.2177741	111.7569254	111.7569254
b4u	24108.73	33745.15	1	466	triangle	3	107.6936	ok	466;180;618	0.6822;0.0	107.3954934	107.7922503	107.7922503
b5u	23439.75	32630.19	1	614	triangle	3	104.4032	ok	614;461;615	0.8619;0.0	103.866722	105.1582841	104.3025338
b6u	23567.18	31371.88	1	302	triangle	3	101.7219	ok	302;28;413	0.3396;0.0	101.6057541	101.8011708	101.6057541
b7u	21910.66	30591.41	1	658	triangle	3	99.495	ok	658;367;442	0.9203;0.0	98.81247863	100.163798	99.53247077
b8u	22372.58	31276.31	1	930	triangle	3	100.5604	ok	930;113;952	0.8627;0.0	100.0571849	100.9350965	100.5305706
b9u	21289.47	32168.28	1	503	triangle	3	103.698	ok	503;228;560	0.1884;0.2	103.4707395	103.8930508	103.4707395
b10u	22165.51	33108.03	1	973	triangle	3	104.7663	ok	973;896;983	0.4978;0.0	104.5612466	105.6972992	104.8999766
b11u	23439.75	33665.51	1	822	triangle	3	106.9897	ok	822;617;847	0.1302;0.1	106.782551	107.0456988	106.782551
b12u	21130.19	34653.05	1	491	triangle	3	109.9303	ok	491;343;593	0.6286;0.0	109.0581019	111.1116056	110.2142695
b13u	22930.06	37392.67	1	330	triangle	3	118.3606	ok	330;58;198	0.9423;0.0	118.3076588	120	118.3076588
b14u	23216.76	34637.12	1	833	triangle	3	109.5328	ok	833;622;732	0.8217;0.0	109.3528456	110.7316048	109.3528456
b15u	22277.01	34318.56	1	820	triangle	3	108.3128	ok	820;597;904	0.5431;0.2	107.7220115	108.5667891	108.5667891
b1l	21353.18	35943.22	2	487	triangle	3	113.5646	ok	487;339;484	0.7246;0.1	113.2450978	114.5740584	113.2450978
b2l	22085.87	35560.94	2	853	triangle	3	112.5255	ok	853;636;852	0.5300;0.0	112.2573686	113.1937691	112.2573686
b3l	23630.89	35258.31	2	471	triangle	3	111.5174	ok	471;322;582	0.1249;0.1	111.2184051	111.7577925	111.7577925
b4l	24108.73	33745.15	2	466	triangle	3	107.6921	ok	466;180;618	0.6822;0.0	107.3931796	107.7905692	107.7905692
b5l	23439.75	32630.19	2	614	triangle	3	104.4042	ok	614;461;615	0.8619;0.0	103.8661262	105.1574567	104.3037555
b6l	23567.18	31371.88	2	302	triangle	3	101.7225	ok	302;28;413	0.3396;0.0	101.6060695	101.801914	101.6060695
b7l	21910.66	30591.41	2	658	triangle	3	99.52019	ok	658;367;442	0.9203;0.0	98.89685207	100.1718225	99.55375842
b8l	22372.58	31276.31	2	930	triangle	3	100.643	ok	930;113;952	0.8627;0.0	100.191629	101.0152294	100.6117596
b9l	21289.47	32168.28	2	503	triangle	3	103.6972	ok	503;228;560	0.1884;0.2	103.4699393	103.891484	103.4699393
b10l	22165.51	33108.03	2	973	triangle	3	104.8147	ok	973;896;983	0.4978;0.0	104.6238469	105.7039062	104.9377125
b11l	23439.75	33665.51	2	822	triangle	3	106.9921	ok	822;617;847	0.1302;0.1	106.7837092	107.0489752	106.7837092
b12l	21130.19	34653.05	2	491	triangle	3	109.9302	ok	491;343;593	0.6286;0.0	109.0573614	111.1123209	110.2143698
b13l	22930.06	37392.67	2	330	triangle	3	118.2962	ok	330;58;198	0.9423;0.0	118.2487001	119.6147132	118.2487001
b14l	23216.76	34637.12	2	833	triangle	3	109.5349	ok	833;622;732	0.8217;0.0	109.3552857	110.7323914	109.3552857
b15l	22277.01	34318.56	2	820	triangle	3	108.3501	ok	820;597;904	0.5431;0.2	107.8120285	108.5914938	108.5914938

Interpolating from a Voronoi Grid

Then Claude compared the two SSF files

Excellent agreement. Across all 30 bores the two methods match to **max 0.022 m, mean 0.005 m, RMS 0.008 m**, on heads spanning ~99-118 m — i.e. differences of a few millimetres to at most ~2 cm. That confirms my Python barycentric interpolation reproduces MF6INTFAC/MF6MOD2SMP.

The small residuals are just triangle choice: where a bore sits among several connected cell centres, MF6INTFAC may form a slightly different enclosing triangle (or fall back to a quad/polygon) than my "first enclosing triangle" pick, giving marginally different weights. Let me confirm that's the source by checking how many nodes gwutil used per bore.

That settles it. Both methods used exactly 3 nodes — a barycentric triangle — for every bore, so there's no quad/IDW fallback involved. The residuals come purely from *which* triangle of connected cell centres each implementation selects: where a bore has several valid enclosing triangles among its neighbours, MF6INTFAC and my code sometimes pick a different trio, producing slightly different weights (e.g. b15, b3). The effect is millimetre-to-2cm level and physically negligible.

Bottom line: the two approaches agree to within ~0.008 m RMS across all 30 bores. Your Python result (`bores_heads.smp`) and the gwutil result (`bores_mf6.smp`) are equivalent, and either is fine to use — the gwutil `.smp` is the one to feed downstream utilities (SMP2HYD, PESTPREP, etc.) if this ever goes into a PEST workflow.

I can only assume that CLAUDE read the BLN file written by MF6INTFAC. How smart is that?

If you want them to match to the last digit, I can change my triangle selection to mirror MF6INTFAC's exact construction rule, but given the agreement that's optional. Anything else on this model?

Interpolating from a Voronoi Grid

The big question

I have checked and checked and checked and modified MF6INTFAC to ensure that it works in complex situations, for example:

- more complex polygons
- triangular cells instead of polygons
- partially missing layers (can affect identification of horizontally connected cells)

Would Claude work in these difficult situations?

Once I would have doubted it.

Now I am not so sure!



Claude doesn't need any help from me

**Generating Nonstationary Stochastic Fields
that
Populate a Voronoi Grid**

Generating Nonstationary Stochastic Fields

Workplan

- Use FIELDGEN2D_SVA from the PEST Groundwater Utility Suite to generate the stochastic fields.
- FIELDGEN2D_SVA requires a “2D node specification file”. Ask Claude to write this.
- FIELDGEN2D_SVA requires anisotropic spatial interpolation of geostatistical hyperparameters ascribed to “conceptual points” to every cell in the model domain. Use PLPROC to do this, but ask Claude to write the PLPROC script.
- Ask Claude to assist in 2D and 3D display of stochastic fields.

Making a Node Specification File

I asked Claude

I would like to build a "2d node specification file". This is described in documentation for FIELDGEN2D_SVA which is discussed in *gwutil_b.pdf*. I would like this node specification file to be named *node.spc*.

A "node specification file" lists model cell locations and dimensions.

From *gwutil_b.pdf*
(i.e. one of the manuals for the PEST Groundwater Utilities)

node	x	y	area	active
1	466082.499	7490090.611	62500	0
2	466294.511	7489958.131	62500	0
3	466506.523	7489825.651	62500	0
4	466718.535	7489693.172	62500	0
5	466930.547	7489560.692	62500	0
6	467142.559	7489428.212	62500	0
7	467354.571	7489295.732	62500	0
8	467566.583	7489163.252	62500	0
9	467778.595	7489030.772	62500	0
10	467990.607	7488898.293	15625	0
11	468202.619	7488765.813	15625	0
12	468414.631	7488633.333	15625	0
13	468626.643	7488500.853	15625	0
14	468838.655	7488368.373	15625	0

First few lines of a 2D node specification file.

A problem (which Claude pointed out to me)

In the manual, this is an image and not text.

So Claude created a text file based on the image.

Making a Node Specification File

Claude said

Built `geostats/node.spc` following the `FIELDGEN2D_SVA` spec in `gwutil_b.pdf` (p73): five columns with the exact required headers `node x y area active`. It covers the 2D per-CPL voronoi layer — 1009 nodes numbered sequentially from 1, with cell-centre x/y (from `CELLX/CELLY` in the `grb`), each cell's polygon area, and `active=1` throughout (`IDOMAIN` is 1 everywhere). Cross-checks pass: nodes sequential, all areas positive, and they sum to exactly 30,000,000 m².

I also saved `make_node_spc.py` so it can be regenerated if the grid changes.

To actually run `FIELDGEN2D_SVA` you'll also need the companion "2D averaging function specification file" (node plus mean, sill/variance, aa/a, bearing, anisotropy). Want me to build a template for that next?

File `node.spc` (generated by Claude)

node	x	y	area	active
1	21512.499	30043.805	5160.952	1
2	21603.752	30027.946	5457.883	1
3	21705.491	30025.941	5008.532	1
4	21805.655	30026.204	5055.994	1
5	21905.291	30025.969	4997.754	1
6	22005.046	30025.675	4941.852	1
7	22104.999	30026.206	5041.842	1
8	22204.807	30026.932	5194.263	1
9	22304.807	30026.598	5153.220	1
10	22409.287	30026.406	5333.495	1
11	22500.000	30015.556	2777.774	1
12	22596.487	30035.172	7762.821	1
13	22707.627	30025.797	4955.850	1
14	22807.382	30026.068	5025.123	1

etc

So now `FIELDGEN2D_SVA` knows the target cells for the stochastic fields that it must generate.

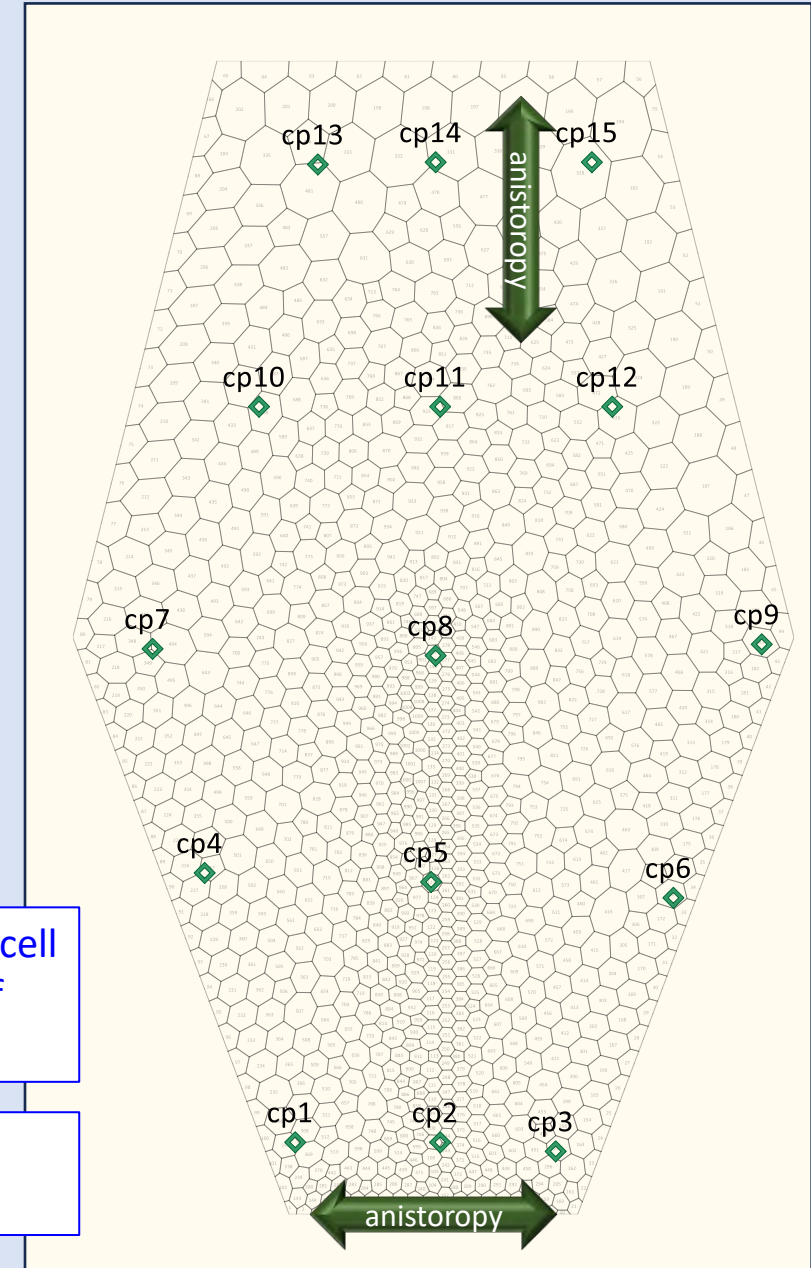
Interpolating from Conceptual Points

Geostatistical hyperparameters ascribed to each conceptual point
(encapsulated in a "conceptual points file" written by me)

point	easting	northing	mean	sill	aa	anis	bearing
cp1	21542.446782096	30499.436596842	4.0	0.2	2500	2.7	90
cp2	22547.092781208	30499.436596842	4.0	0.2	2500	2.7	90
cp3	23347.670061751	30436.646221898	4.0	0.2	2500	2.7	90
cp4	20914.543032651	32367.450251442	3.0	0.1	1600	1.8	90
cp5	22484.302406264	32304.659876498	3.0	0.1	1600	1.8	90
cp6	24163.94493603	32194.776720345	3.0	0.1	1600	1.8	90
cp7	20553.498376719	33921.512031319	2.5	0.15	1600	1.0	90
cp8	22515.697593736	33874.419250111	2.5	0.15	1600	1.0	90
cp9	24776.151091739	33952.907218792	2.5	0.15	1600	3.0	0
cp10	21291.285282318	35601.154561086	2.0	0.15	1500	3.0	0
cp11	22547.092781208	35601.154561086	2.0	0.15	1500	3.0	0
cp12	23740.109905155	35601.154561086	2.0	0.15	1500	3.0	0
cp13	21699.422719457	37280.797090852	2.0	0.15	1500	3.0	0
cp14	22515.697593736	37296.494684588	2.0	0.15	1500	3.0	0
cp15	23598.831561529	37296.494684588	2.0	0.15	1500	3.0	0

Geostatistical hyperparameters must be spatially interpolated from conceptual points to every cell in each model layer. (Hyperparameter values are different for different model layers because of differing geology.)

However, interpolation must be anisotropic in order to respect the anisotropy that the hyperparameters describe.



Interpolating from Conceptual Points

I asked Claude

I normally do this kind of interpolation using PLPROC. I interpolate twice. On the second occasion, interpolation is anisotropic because I have interpolated the hyperparameters to the model grid in the first interpolation to use in the second interpolation.

I gave Claude the PLPROC manual to read. I told Claude about the PLPROC functions that I use for this kind of workflow. These are *calc_kriging_factors_2d()*, *calc_kriging_factors_auto_2d()*, *calc_kriging_factors_2d_SVA()*.

I asked Claude whether it had any better ideas?

Claude said

I wouldn't reimplement SVA kriging in Python: PLPROC is the reference implementation, you already trust it, and I couldn't validate a hand-rolled version against anything better.

So I asked Claude

Please write the entire PLPROC script file. The script must inform PLPROC to write the outcomes of the interpolation process in a file that can be read by FIELDGEN2D_SVA as its "averaging function specification file". As you can see from FIELDGEN2D_SVA documentation, this has the same format as that which I provided to you in the conceptual points file. However the "points" in this case are every cell in the model grid.

Great. So I'm not completely replaceable by Claude yet. (At least until next month maybe.)

Interpolating from Conceptual Points

PLPROC script file *make_avefunc.plp* written by Claude

```
# =====
# make_avgfunc.plp
#
# Build the "2D averaging function specification file" required by
# FIELDGEN2D_SVA, by interpolating the geostatistical hyperparameters
# (mean, sill, aa, anis, bearing) from the conceptual points in cpl.dat to
# every voronoi cell of the DISV grid (NCPL = 1009).
#
# Two-stage interpolation so the interpolation itself honours the anisotropy
# that the hyperparameters describe:
#
# STAGE 1 isotropic ordinary kriging (calc_kriging_factors_2d) with an
# EXPONENTIAL variogram and a LARGE range (a = 3000 m) -> a smooth
# first-pass anisotropy field (anis, bearing) at every cell.
#
# STAGE 2 SVA kriging (calc_kriging_factors_sva_2d) with an EXPONENTIAL
# variogram, steered by the stage-1 anis / bearing fields
# (anis_ratio_plist, anis_bearing_plist) and enforce_strong_anis.
#
# KEY SMOOTHNESS KNOB -- INTERP_A (below):
# The variogram range used to INTERPOLATE the hyperparameters is set to a
# large constant (3000 m), DECOUPLED from the property correlation length
# 'aa' (~1000 m). Interpolating the hyperparameters with the short property
# 'aa' is what pinched the field into bull's-eyes around the conceptual
# points; a range at the scale of the conceptual-point spacing removes them.
# Changing INTERP_A does NOT change the reported aa/anis/etc values -- it
# only affects how smoothly they are spread between points. Reduce it
# (e.g. 1500-2000) if the fields look too smooth; raise it for smoother.
#
# Interpolation is LOGARITHMIC for mean, sill, aa and anis (all strictly
# positive), and LINEAR for bearing (a direction in degrees).
#
# Run on Windows in the voro\geostats folder: plproc make_avgfunc.plp
# Output: avgfunc.dat columns: node mean sill aa anis bearing
# =====

#-- 1. TARGET grid : 2D CLIST of the 1009 per-CPL voronoi cell centres -----
cl_grid = read_mf6_grid_specs(file='..\model.disv.grb', &
                             dimensions=2)

#-- 2. SOURCE : the 15 conceptual points in cpl.dat -----
# col1 = point (char id) col2 = easting (x) col3 = northing (y)
# col4 = mean col5 = sill col6 = aa col7 = anis col8 = bearing
cl_cp = read_list_file(file='cpl.dat',
                      &
                      dimensions=2,
                      id_type='character',
                      skiplines=1,
                      plist='mean_cp';column=4,
                      plist='sill_cp';column=5,
                      plist='aa_cp';column=6,
                      plist='anis_cp';column=7,
                      plist='bear_cp';column=8)
```

```
# STAGE 1 : isotropic exponential kriging, large range -> anisotropy field
# -----
anis_1 = new_plist(reference_clist=cl_grid,value=0.0)
bear_1 = new_plist(reference_clist=cl_grid,value=0.0)

calc_kriging_factors_2d(target_clist=cl_grid,
                       source_clist=cl_cp,
                       file='fac_stagel.dat';form='text',
                       variogram='exponential',
                       a=3000,
                       kriging='ordinary',
                       min_points=1,max_points=15,search_radius=1e15)

# anis is logarithmic; bearing is linear
anis_1 = anis_cp.krige_using_file(file='fac_stagel.dat';form='text',transform='log')
bear_1 = bear_cp.krige_using_file(file='fac_stagel.dat';form='text',transform='none')

# anisotropy ratio must be >= 1.0 before it is used to steer stage 2
anis_1(select=(anis_1 < 1.0)) = 1.0

# -----
# STAGE 2 : SVA exponential kriging, steered by stage-1 anisotropy field
# interpolation range = INTERP_A (constant, decoupled from aa)
# -----
# Final grid PLISTS are named exactly as FIELDGEN2D_SVA expects its headers.
mean = new_plist(reference_clist=cl_grid,value=0.0)
sill = new_plist(reference_clist=cl_grid,value=0.0)
aa = new_plist(reference_clist=cl_grid,value=0.0)
anis = new_plist(reference_clist=cl_grid,value=0.0)
bearing = new_plist(reference_clist=cl_grid,value=0.0)

calc_kriging_factors_sva_2d(target_clist=cl_grid,
                            source_clist=cl_cp,
                            file='fac_stage2.dat';form='text', &
                            variogram='exponential',
                            a=3000,
                            kriging='ordinary',
                            anis_ratio_plist=anis_1,
                            anis_bearing_plist=bear_1,
                            enforce_strong_anis='yes',
                            search_radius=1e15,
                            max_points=15,min_points=1)

# logarithmic interpolation for mean / sill / aa / anis; linear for bearing
mean = mean_cp.krige_using_file(file='fac_stage2.dat';form='text',transform='log')
sill = sill_cp.krige_using_file(file='fac_stage2.dat';form='text',transform='log')
aa = aa_cp.krige_using_file(file='fac_stage2.dat';form='text',transform='log')
anis = anis_cp.krige_using_file(file='fac_stage2.dat';form='text',transform='log')
bearing = bear_cp.krige_using_file(file='fac_stage2.dat';form='text',transform='none')

#-- Guard rails on the final fields -----
anis(select=(anis < 1.0)) = 1.0 # anisotropy ratio never below 1.0

# -----
# Export the complete averaging function file: node mean sill aa anis bearing
# -----
write_column_data_file(file='avgfunc.dat';delim='space', &
                      header='yes',
                      clist_spec='id';headtext='node', &
                      plist=mean,
                      plist=sill,
                      plist=aa,
                      plist=anis,
                      plist=bearing)
```

Interpolating from Conceptual Points

Running PLPROC

```
E:> plproc make_avgfunc.plp
```

```
PLPROC Version 5.0. Watermark Numerical Computing.
```

```
Reading and storing contents of PLPROC script file make_avgfunc.plp...  
Processing commands in PLPROC script file...
```

```
> cl_grid=read_mf6_grid_specs(file='..\model.disv.grb',dimensions=2)  
> cl_cp=read_list_file(file='cp1.dat',dimensions=2,id_type='character',...  
> anis_1=new_plist(reference_clist=cl_grid,value=0.0)  
> bear_1=new_plist(reference_clist=cl_grid,value=0.0)  
> calc_kriging_factors_2d(target_clist=cl_grid,source_clist=cl_cp,file...  
> anis_1=anis_cp.krige_using_file(file='fac_stage1.dat';form='text',tra...  
> bear_1=bear_cp.krige_using_file(file='fac_stage1.dat';form='text',tra...  
> anis_1(select=(anis_1<1.0))=1.0  
> mean=new_plist(reference_clist=cl_grid,value=0.0)  
> sill=new_plist(reference_clist=cl_grid,value=0.0)  
> aa=new_plist(reference_clist=cl_grid,value=0.0)  
> anis=new_plist(reference_clist=cl_grid,value=0.0)  
> bearing=new_plist(reference_clist=cl_grid,value=0.0)  
> calc_kriging_factors_sva_2d(target_clist=cl_grid,source_clist=cl_cp,f...  
> mean=mean_cp.krige_using_file(file='fac_stage2.dat';form='text',trans...  
> sill=sill_cp.krige_using_file(file='fac_stage2.dat';form='text',trans...  
> aa=aa_cp.krige_using_file(file='fac_stage2.dat';form='text',transform...  
> anis=anis_cp.krige_using_file(file='fac_stage2.dat';form='text',trans...  
> bearing=bear_cp.krige_using_file(file='fac_stage2.dat';form='text',tr...  
> anis(select=(anis<1.0))=1.0  
> write_column_data_file(file='avgfunc.dat';delim='space',header='yes',...
```

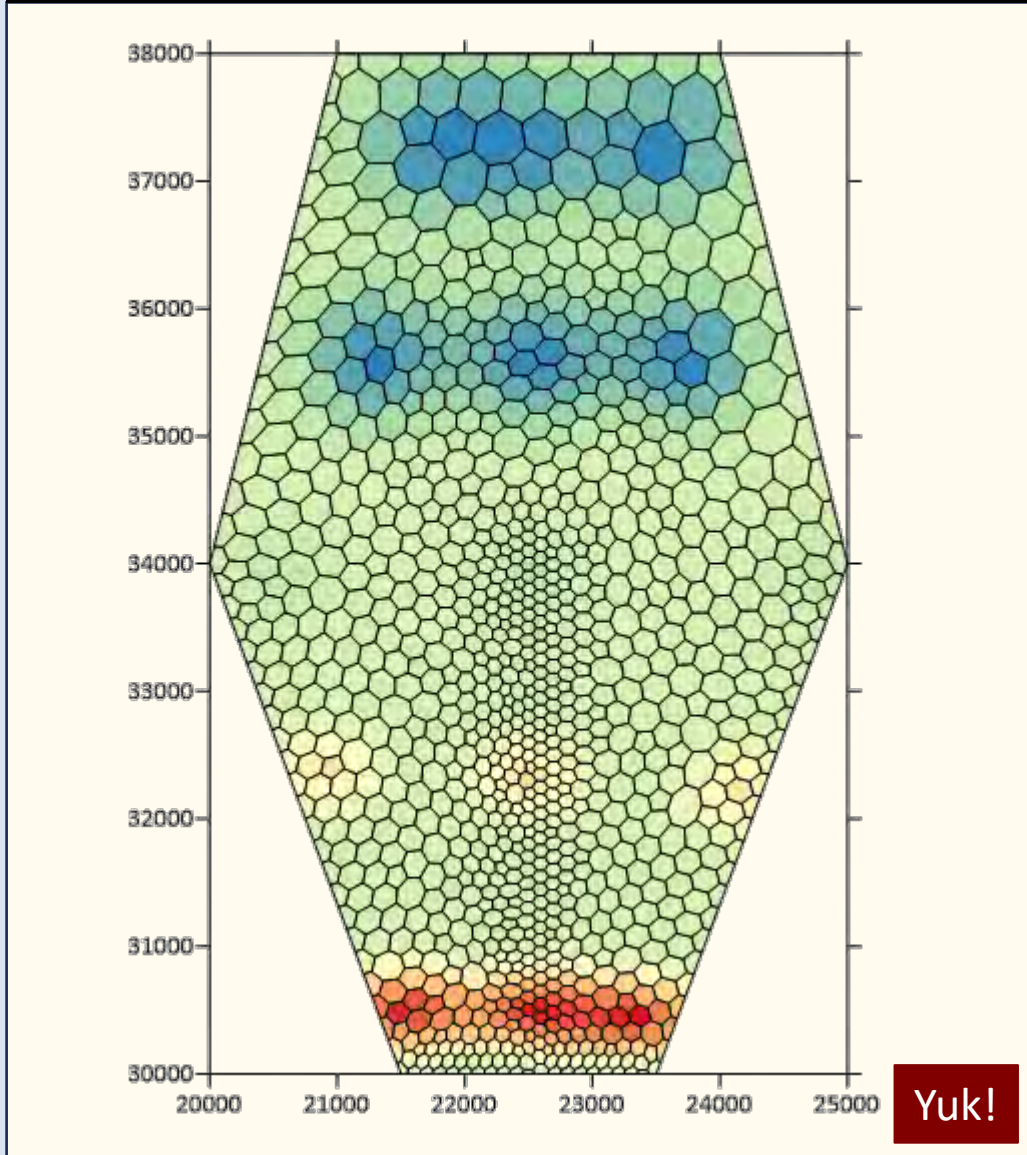
node	mean	sill	aa	anis	bearing
1	3.645985	0.1895765	2313.215	2.636789	79.71858
2	3.640920	0.1895303	2311.014	2.636707	79.53096
3	3.645735	0.1898237	2314.268	2.638791	79.63120
4	3.651786	0.1901419	2318.121	2.641011	79.77259
5	3.656866	0.1904056	2321.343	2.642859	79.89187
6	3.661082	0.1906178	2323.985	2.644344	79.99258
7	3.664981	0.1907977	2326.350	2.645590	80.09050
8	3.667940	0.1909216	2328.086	2.646441	80.16832
9	3.668805	0.1909458	2328.536	2.646614	80.19392
10	3.668513	0.1909042	2328.226	2.646331	80.19376
11	3.657504	0.1904405	2321.747	2.643283	79.89915
12	3.672833	0.1909617	2330.182	2.646617	80.34366
13	3.659711	0.1903669	2322.259	2.642665	80.00511
14	3.654302	0.1900549	2318.679	2.640536	79.88419
etc					

I asked Claude

Write a little program (Python will do) to encapsulate these cell-based geostatistical hyperparameters in a shapefile so that I can import them into a GIS.

Interpolating from Conceptual Points

Interpolated values of one geostatistical hyperparameter



I asked Claude

This interpolation process introduces bulls-eyes. Can I show you the picture? *I also made some suggestions of how the PLPROC script could be modified to avoid this problem.*

Claude said

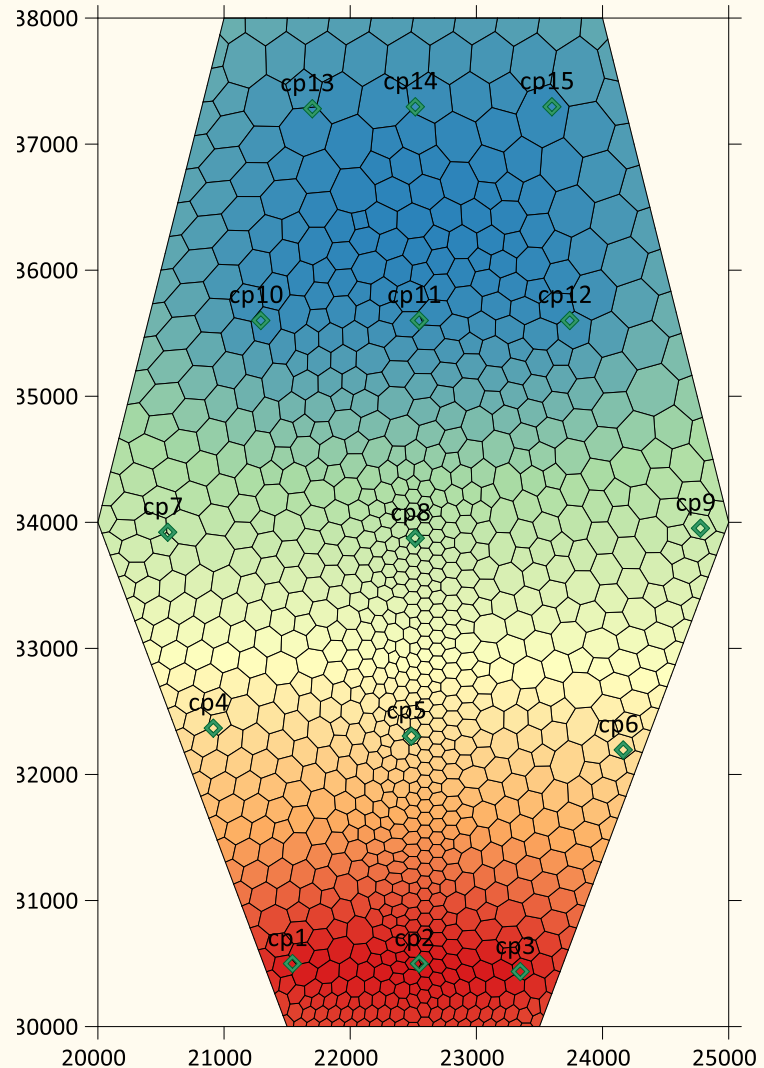
That picture is really useful — it shows exactly the failure mode. The "bull's-eyes" at the conceptual points, with the field relaxing back to a green background in between, are the signature of an interpolation whose variogram range is short relative to the point spacing: ordinary kriging reverts toward the global mean wherever you're more than a range away from a data point.

Here's the key insight about *why* it's happening, which also says whether your fixes will work. Your stage-2 interpolation is using the **property** correlation length `aa` ($\approx 800-1200$ m) as the variogram range for interpolating the **hyperparameters**. But those are two different length scales: `aa` is how the hydraulic property varies (that's for FIELDGEN2D_SVA to use later), whereas the hyperparameters themselves vary smoothly at the scale of your conceptual-point spacing ($\approx 2-3$ km). Interpolating them with a ~ 1 km range is what pinches the field into bull's-eyes.

Claude liked my ideas of how to fix the problem, and came up with some additional ideas of its own.

Generating Stochastic Fields

Much better!!



Now I can run FIELDGEN2D_SVA

```
E:> fieldgen2d_sva
```

```
Program FIELDGEN2D_SVA generates stochastic fields for an arbitrary 2D model  
using an averaging function whose specifications can vary throughout the  
model domain.
```

```
Enter name of 2D node specifications file: node.spc
```

```
- data for 1009 nodes read from file node.spc.
```

```
Enter name of 2D averaging function specification file: avgfunc.dat
```

```
- file avgfunc.dat read ok.
```

```
Express stochasticity in natural or log domain? [n/l]: l
```

```
Is moving ave spherical, exponential, gaussian or power? [s/x/g/p(..)]: g
```

```
How many realizations do you wish to generate? 5
```

```
Enter name for csv output file: fields1.csv
```

```
Enter values for following efficiency-determining variables.
```

```
Summation factor cutoff threshold (<Enter> if 1.0E-5): <Enter>
```

```
Array processing block size (<Enter> if 5): <Enter>
```

```
Restrict averaging to active cells? (y/n) (<Enter> if y): <Enter>
```

```
Enter integer seed for random number generator [324853]: <Enter>
```

```
- undertaking spatial averaging for array block 1...
```

```
- file fields1.csv written ok.
```

Generating Stochastic Fields

CSV file produced by FIELDGEN2D_SVA
(each column attributes simulated values of hydraulic conductivity to all cells within a single model layer)

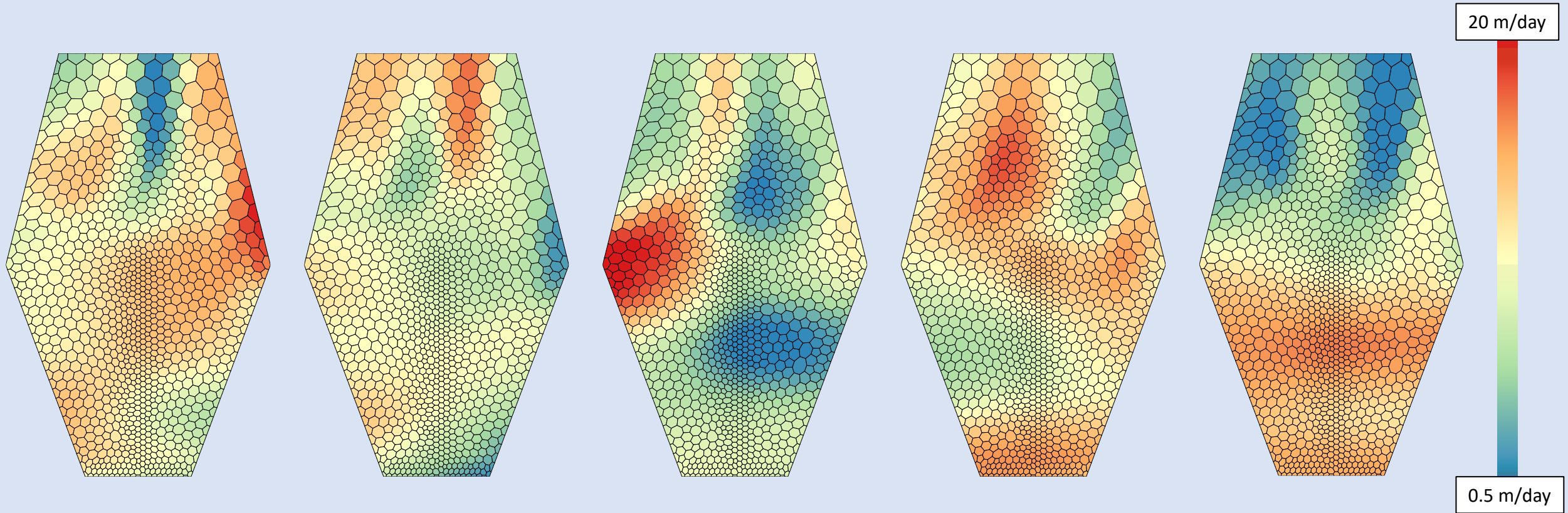
node	sim1	sim2	sim3	sim4	sim5
1	3.521529	1.930087	2.746549	8.879398	7.198481
2	3.318431	1.766134	2.730563	8.834889	7.239814
3	3.172621	1.633308	2.702027	8.80798	7.300137
4	3.052064	1.519558	2.671675	8.774671	7.364222
5	2.942226	1.416022	2.640709	8.726934	7.429365
6	2.842259	1.321686	2.608926	8.666058	7.494695
7	2.753665	1.236904	2.576414	8.596106	7.559512
8	2.673318	1.159753	2.543604	8.515159	7.622735
9	2.596266	1.087228	2.510405	8.416772	7.684308
10	2.523136	1.017991	2.475447	8.301878	7.747242
11	2.437295	0.948587	2.44242	8.127878	7.804215
12	2.430767	0.919529	2.414961	8.120776	7.852608
13	2.349451	0.850388	2.375013	7.903781	7.921099
14	2.303194	0.803353	2.341548	7.751029	7.977453
15	2.261771	0.759733	2.307998	7.585876	8.033317
16	2.224905	0.719165	2.274333	7.405899	8.08926
17	2.193891	0.681854	2.24069	7.214344	8.145392
18	2.169896	0.648097	2.207505	7.016458	8.201306
19	2.151399	0.616409	2.173882	6.793079	8.261754
20	2.139954	0.586698	2.139648	6.510646	8.337505
21	2.143086	0.580123	2.129927	6.591992	8.307226
22	2.184243	0.642025	2.174108	7.430798	7.900312
23	2.234099	0.77271	2.212706	8.066839	7.181813

etc

I asked Claude

Write a little program (in Python if that is easiest for you) that encapsulates all of these stochastic fields in a single shapefile.

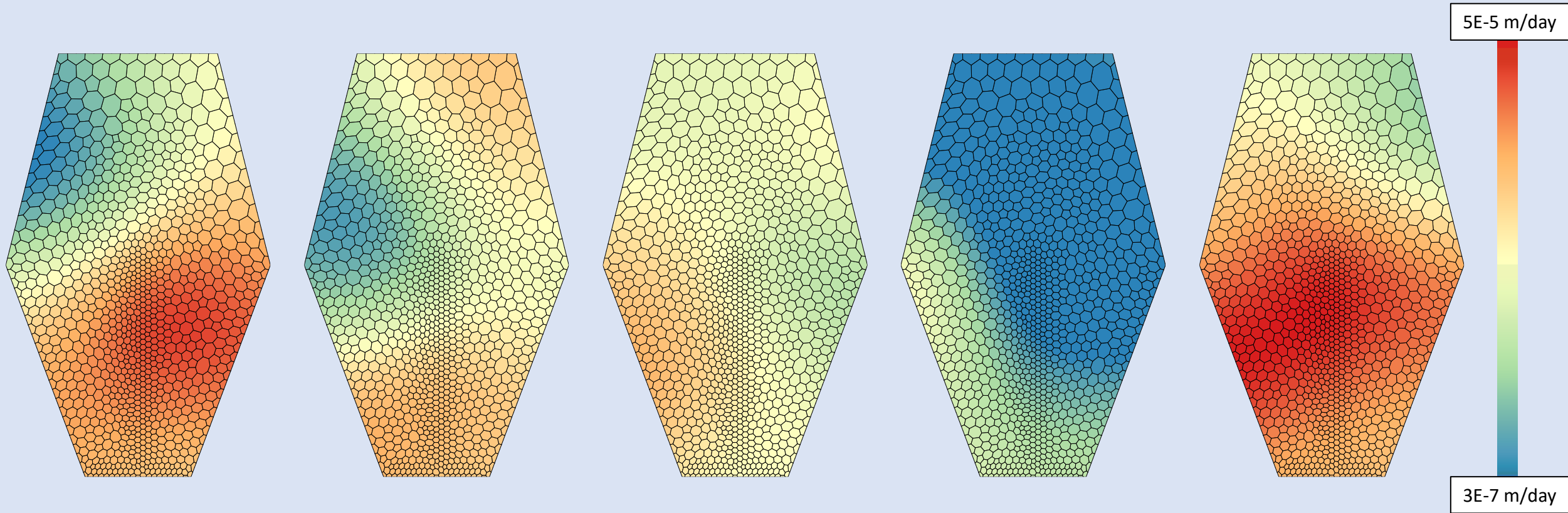
Displaying Stochastic Fields



That was for layer 1

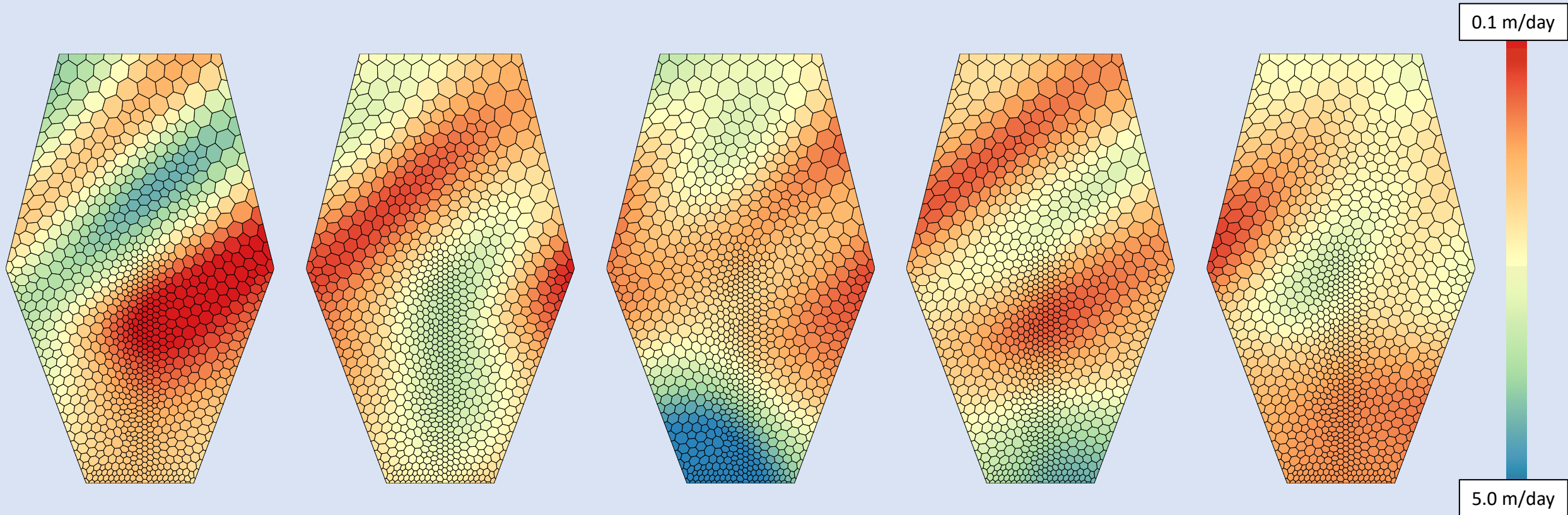
Then I repeated the process, with appropriate geostatistical hyperparameters, for layers 2 and 3.

Displaying Stochastic Fields



Layer 2

Displaying Stochastic Fields



Layer 3

Displaying Stochastic Fields

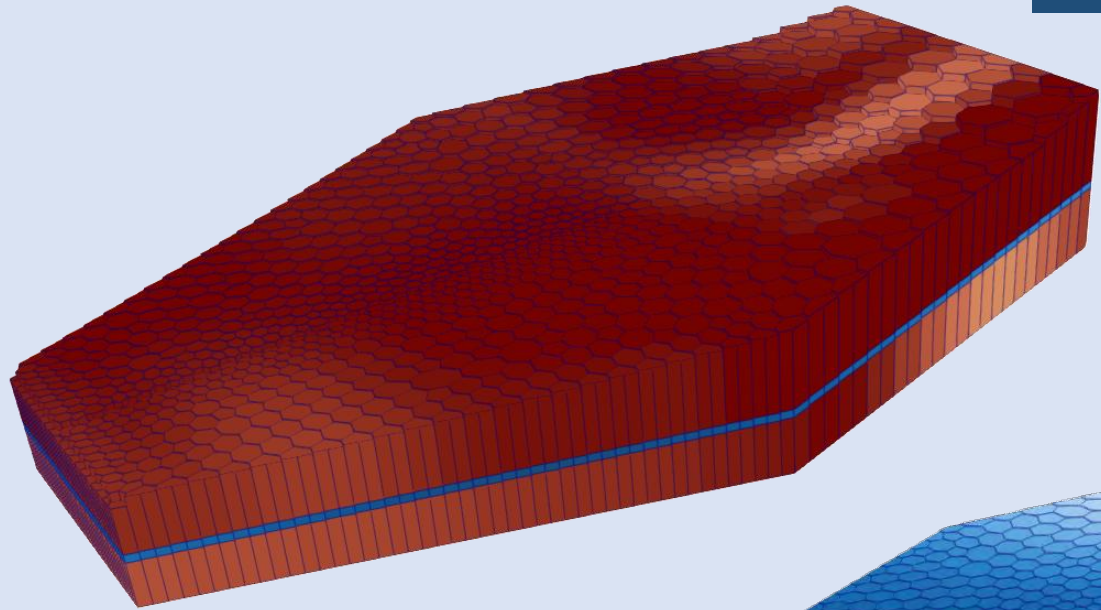
I asked Claude

Files *fields1_lay1.csv*, *fields1_lay2.csv* and *fields1_lay3.csv* each contain 5 stochastic fields. These are for layers 1, 2 and 3 respectively of the MODFLOW 6 model that we have been working with. Create a VTK file so that I can view these 5 stochastic fields for all model layers in PARAVIEW.

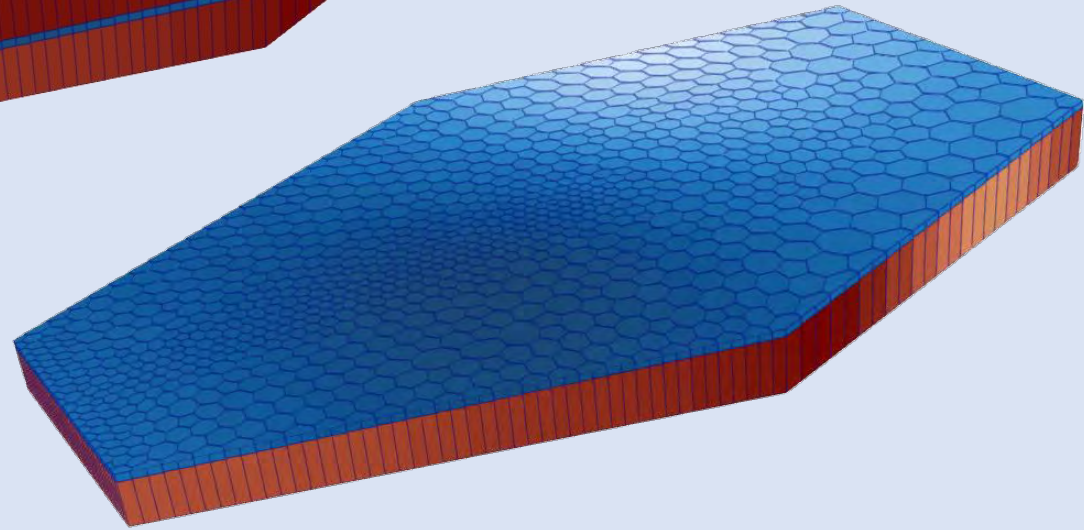
So Claude created and ran a small Python script...

Displaying Stochastic Fields

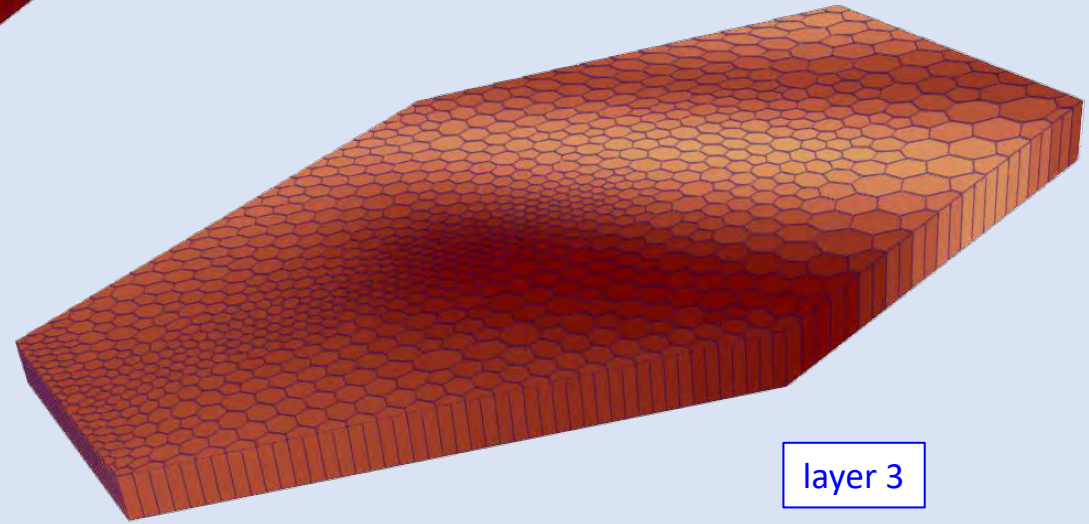
layers 1, 2 and 3



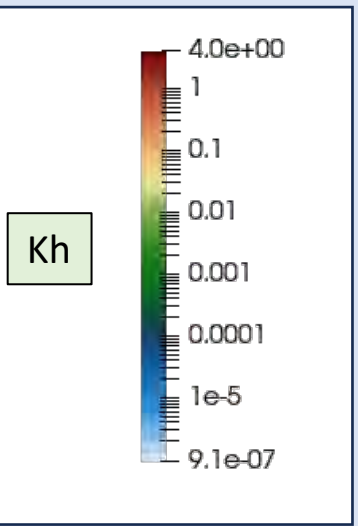
layers 2 and 3



layer 3

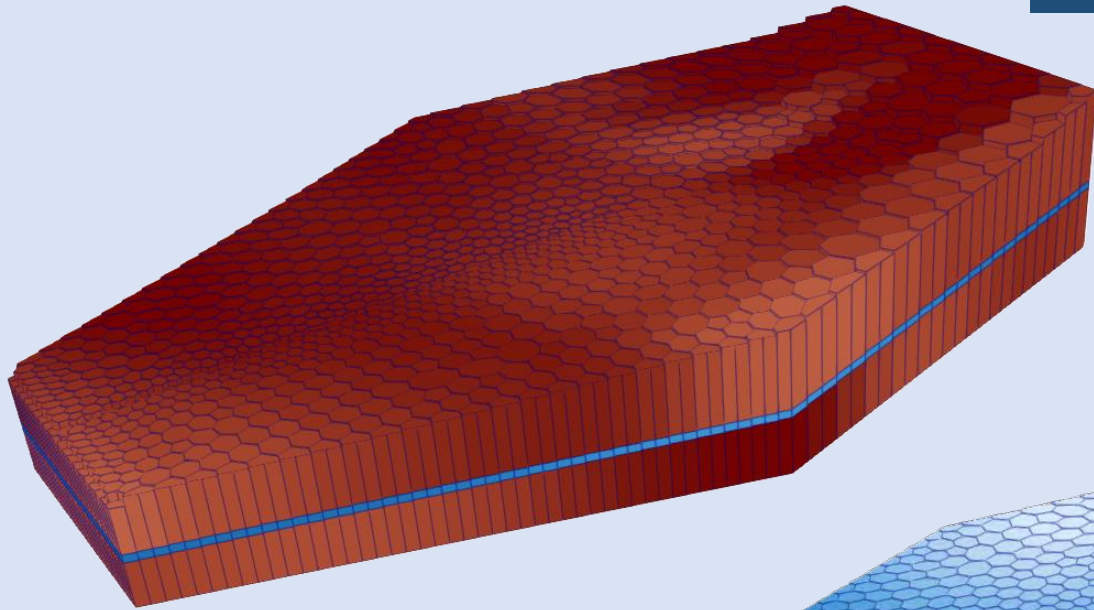


realization #1

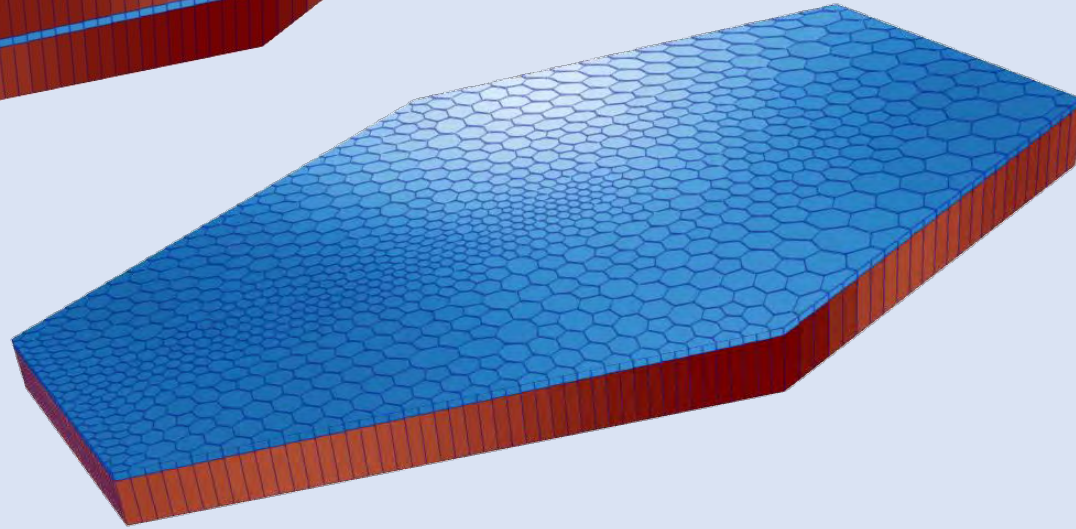


Displaying Stochastic Fields

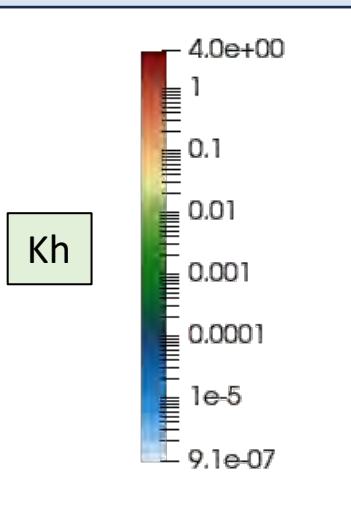
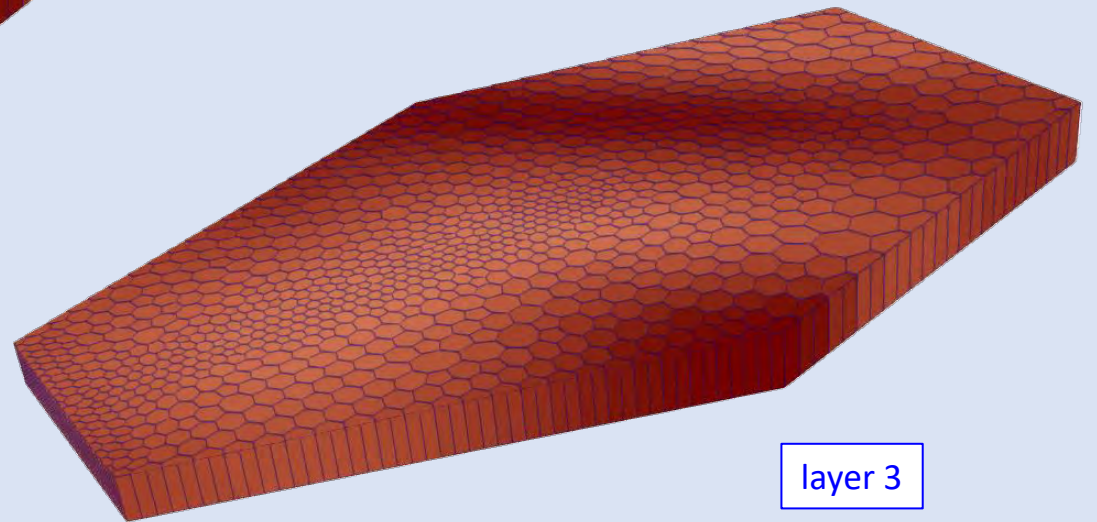
layers 1, 2 and 3



layers 2 and 3



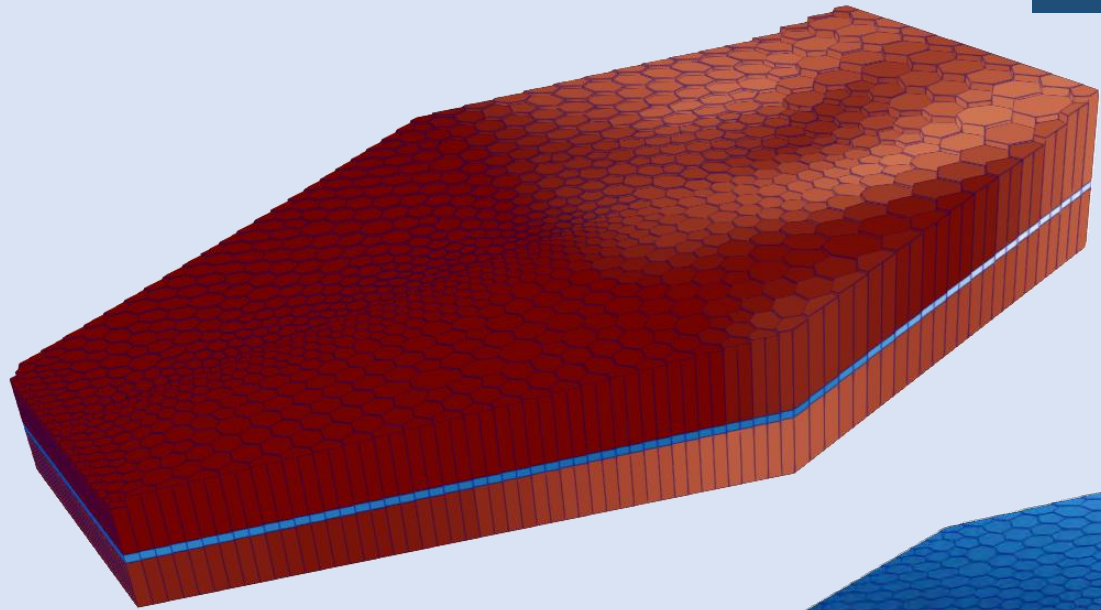
layer 3



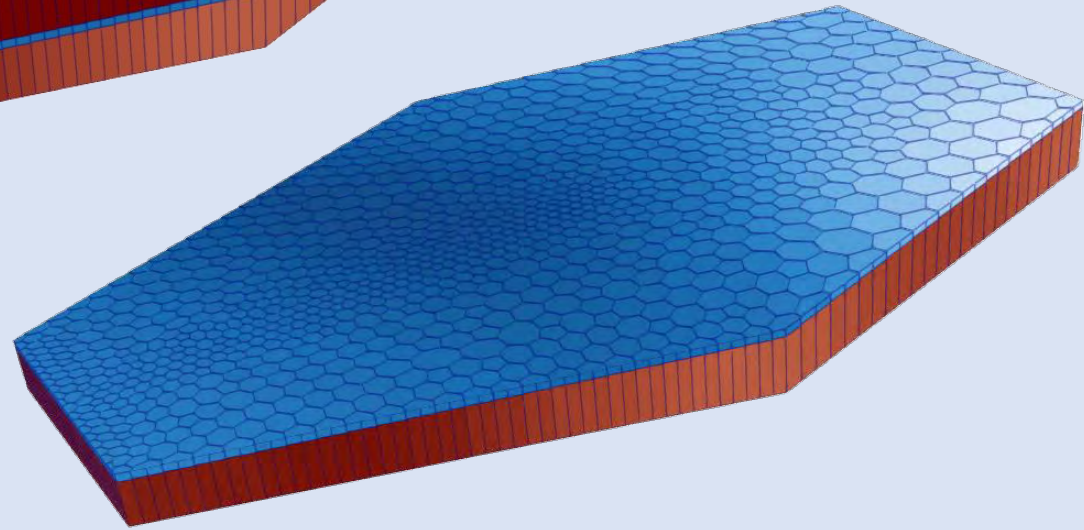
realization #2

Displaying Stochastic Fields

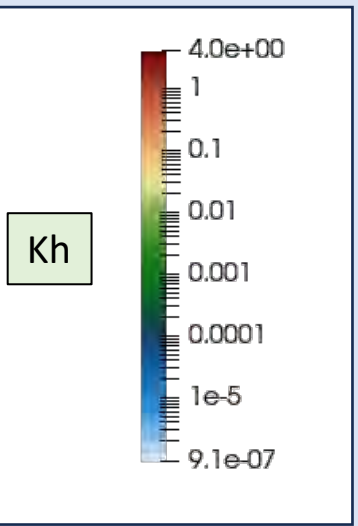
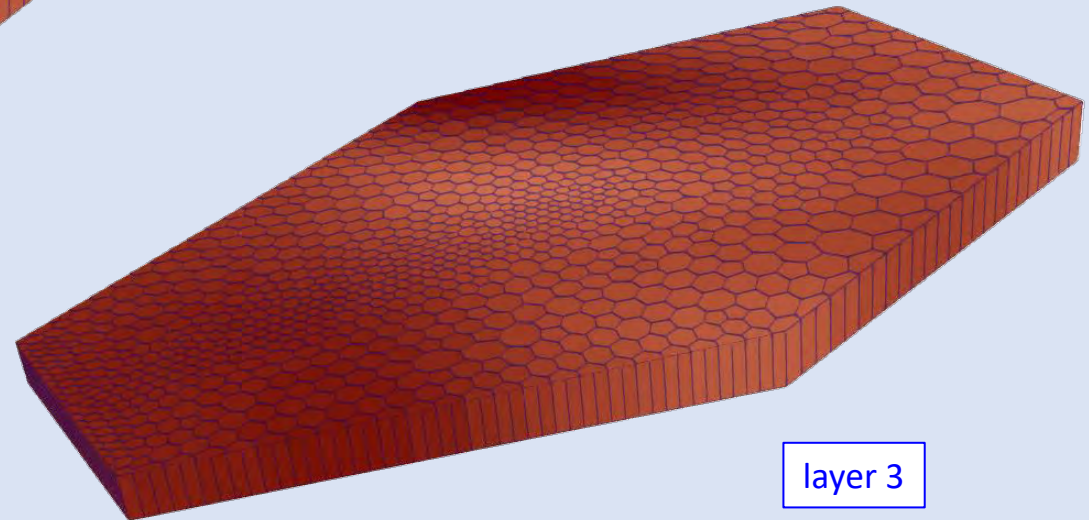
layers 1, 2 and 3



layers 2 and 3



layer 3



realization #5

Adding Functionality to the MODFLOW 6 Model

Recording flow into the central drain

I asked Claude

Please reference an OBS file in the DRN package input file that tells MODFLOW 6 to record flow to all drain cells. Refer to your MODFLOW skills if you need to. Also refer to *mf6io.pdf* (MODFLOW 6 input/output protocols) if you need to.

Modified *model.drn*

```
BEGIN OPTIONS
  BOUNDNAMES
  OBS6 FILEIN model.drn.obs
END OPTIONS

BEGIN DIMENSIONS
  MAXBOUND 40
END DIMENSIONS

BEGIN PERIOD 1
# layer icell2d elevation conductance boundname
  1 103 98.000000 1.0E3 alldrn
  1 104 98.228041 1.0E3 alldrn
  1 105 98.417538 1.0E3 alldrn
  1 106 98.616846 1.0E3 alldrn
  1 107 98.831541 1.0E3 alldrn
  1 108 99.034741 1.0E3 alldrn
  1 109 99.234183 1.0E3 alldrn
etc
```

File *model.drn.obs* (written by Claude)

```
BEGIN OPTIONS
  DIGITS 10
  PRINT_INPUT
END OPTIONS

BEGIN CONTINUOUS FILEOUT model.drn.obs.csv
# obsname obstype id (boundname)
  drn_total drn alldrn
END CONTINUOUS
```

Claude said

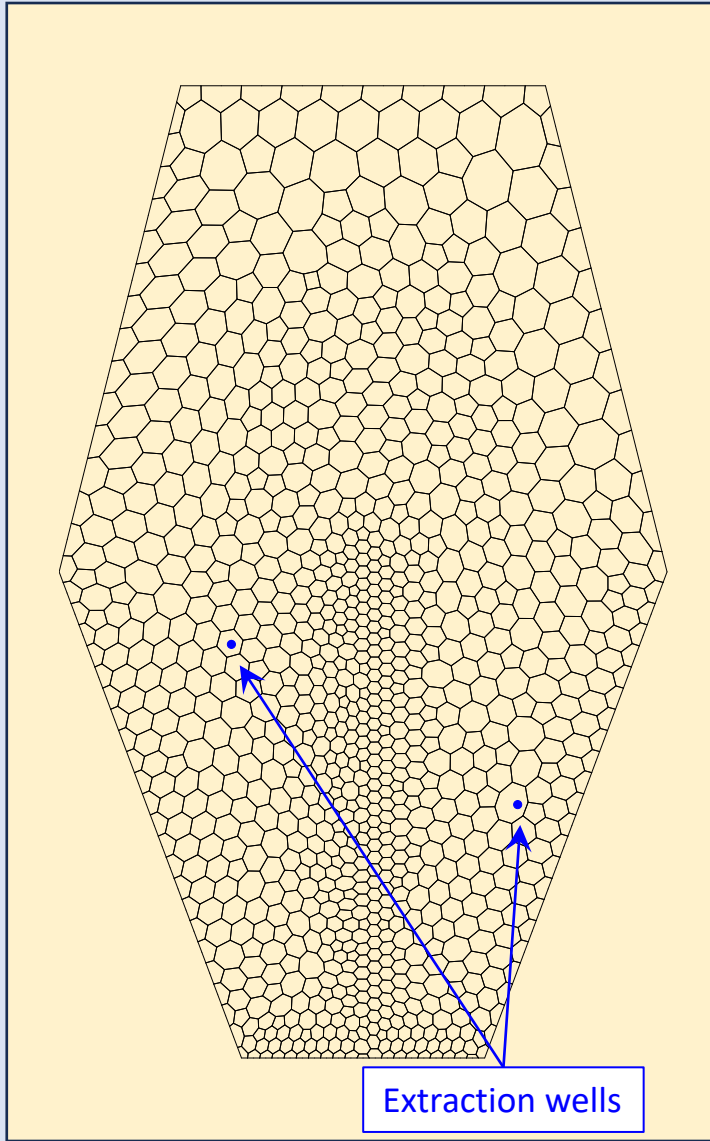
Done. Three things now wire up total drain-flow recording:

`model.drn` OPTIONS block gained `BOUNDNAMES` and `OBS6 FILEIN model.drn.obs`, and each of the 40 drain cells is tagged with the shared boundname `alldrn` (values otherwise unchanged).

`model.drn.obs` (new) defines a single continuous observation, `drn_total`, of type `drn` keyed by the boundname `alldrn`. Because the id is a boundname, MODFLOW 6 sums the flow across every drain cell sharing it, and writes the time series to `model.drn.obs.csv` at run time (sign convention: negative = flow out of into the drains).

the DRN package already sits in `model.nam`, so no name-file change is the OBS file is referenced from within the DRN file itself, which is the pattern. And since MF6 isn't in this sandbox, this hasn't been run; it'll on your next Windows run. Want me to record this in `CLAUDE.md`?

Adding a second stress period and well extraction



File *wells.crd*

```
well1 21416.86, 33403.49  
well2 23771.50, 32084.89
```

Adding a second stress period and well extraction

I asked Claude

See file *wells.crd* that contains the coordinates of two pumping wells (in the second and third columns of this file). These must pump from the third model layer during the second stress period. Locate these wells in the model grid and alter the MODFLOW 6 input dataset to accommodate their presence.

File *model.wel* (written by Claude)

```
BEGIN OPTIONS
  BOUNDNAMES
END OPTIONS

BEGIN DIMENSIONS
  MAXBOUND 2
END DIMENSIONS

BEGIN PERIOD 1
END PERIOD 1

BEGIN PERIOD 2
# layer icell2d  q      boundname
  3  777  -20.0  well1
  3  416  -20.0  well2
END PERIOD 2
```

Claude said

Ran a command, read a file

Coordinates are in model coordinates (well1 ≈ 21417, 33403; well2 ≈ 23772, 32085). Let me locate which voronoi cell each falls in using the grid geometry.

Ran a command

Both wells sit cleanly inside voronoi cells: well1 → icell2d 777, well2 → icell2d 416 (layer 3 cellids 3 777 and 3 416). Now applying the changes.

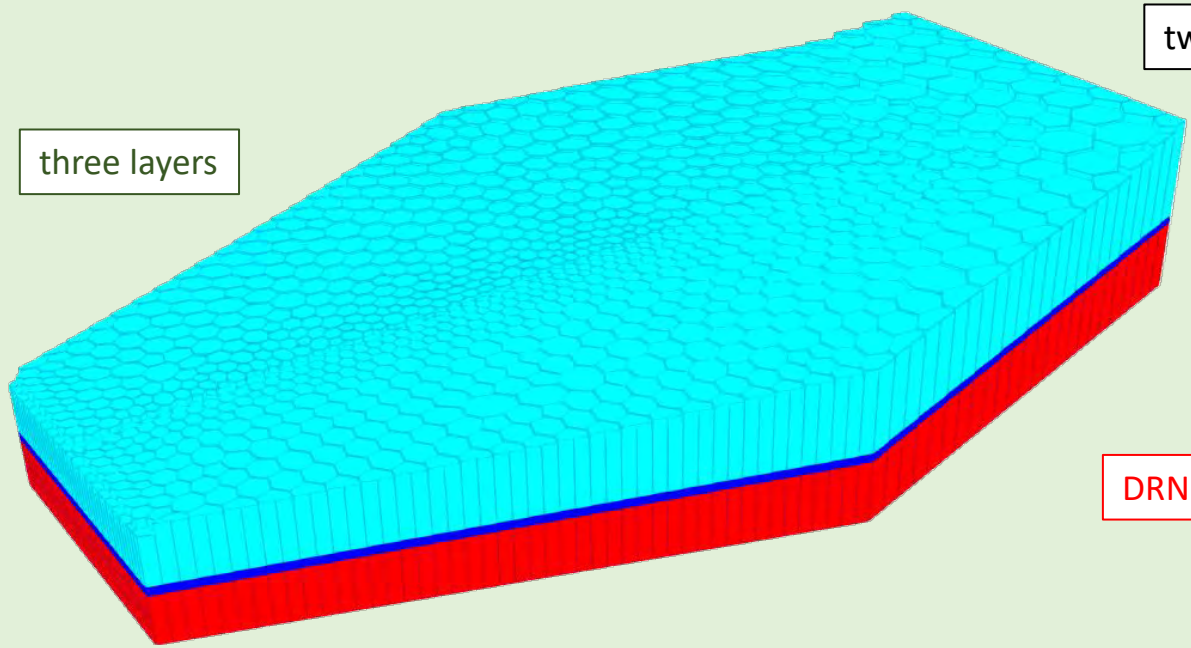
Edited 3 files, read a file, created a file

Done. Summary of changes to the model:

Data Space Inversion

The model

(see previous sections of this slideshow)



two steady-state stress periods

By the way.
The VTK file on which this picture is based was written by Claude.



GHB: head= 98 m

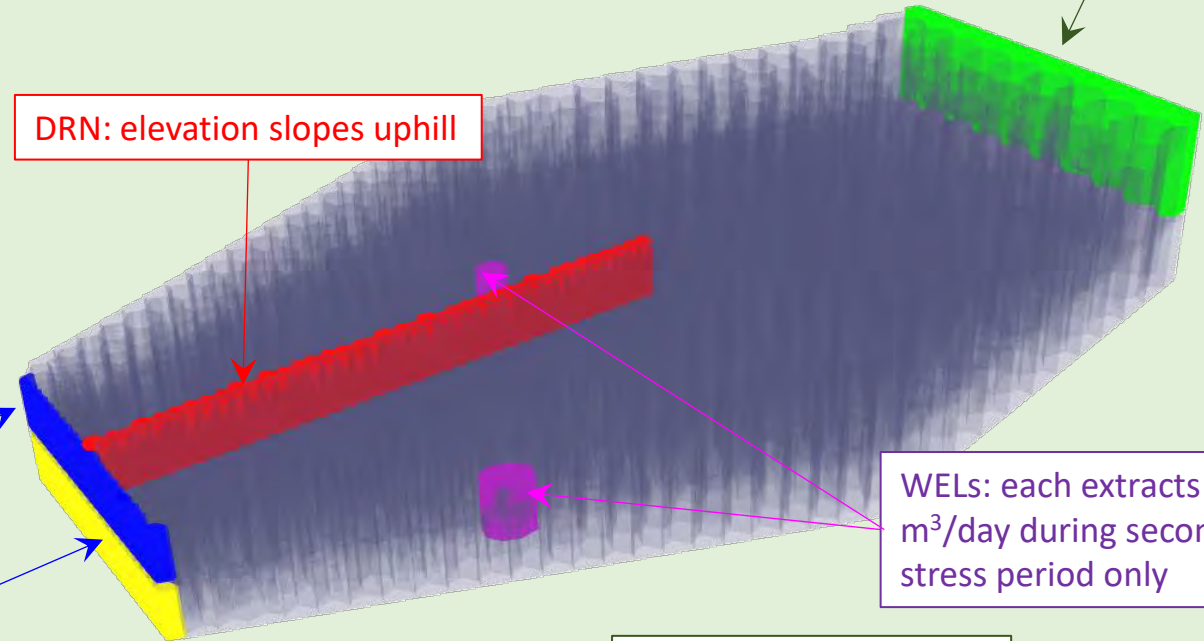
GHB: head= 70 m

DRN: elevation slopes uphill

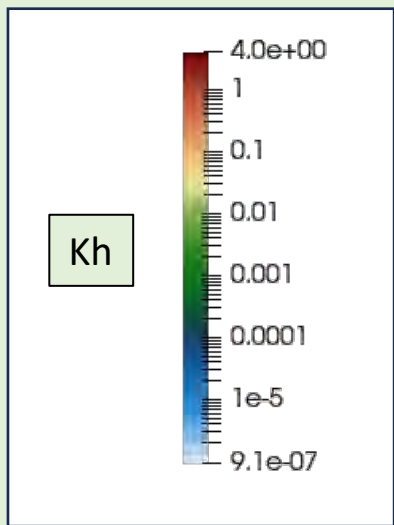
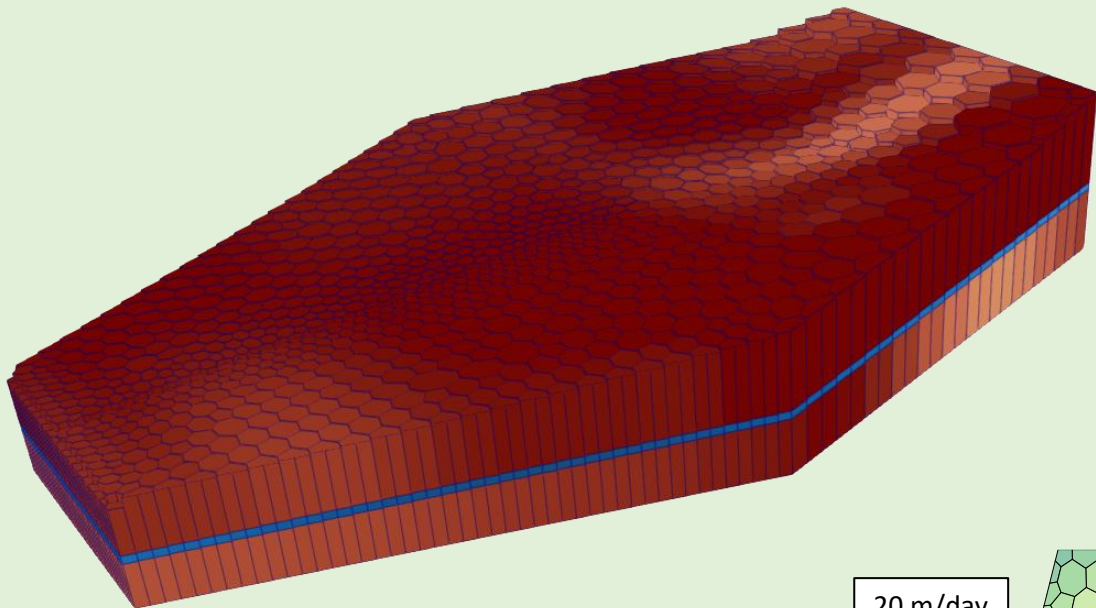
CHD: head= 120 m

WELs: each extracts 20 m³/day during second stress period only

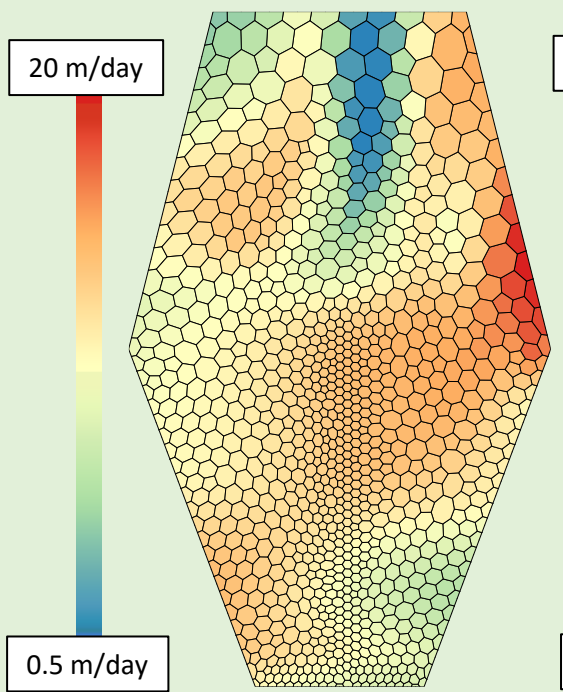
boundary conditions



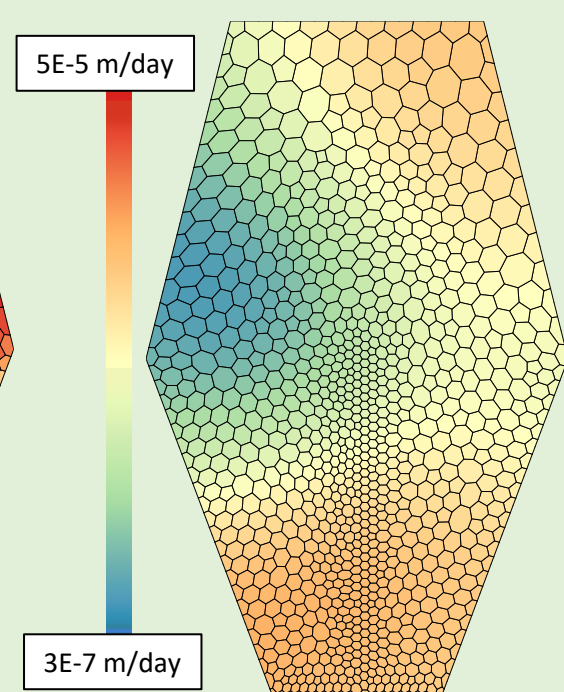
“Reality” hydraulic conductivity



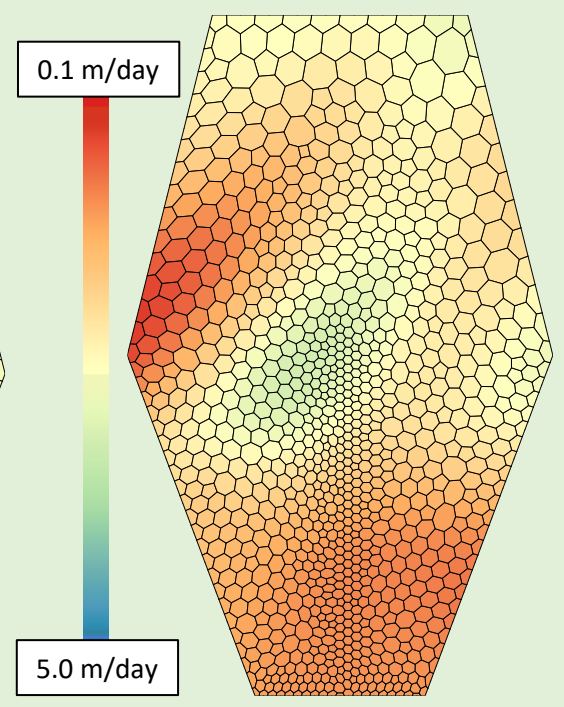
Layer 1



Layer 2



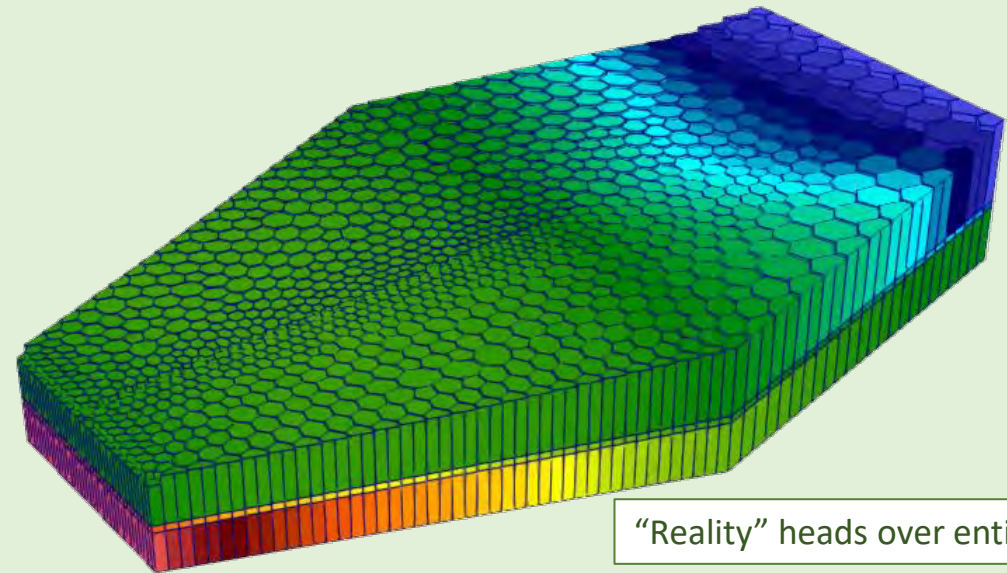
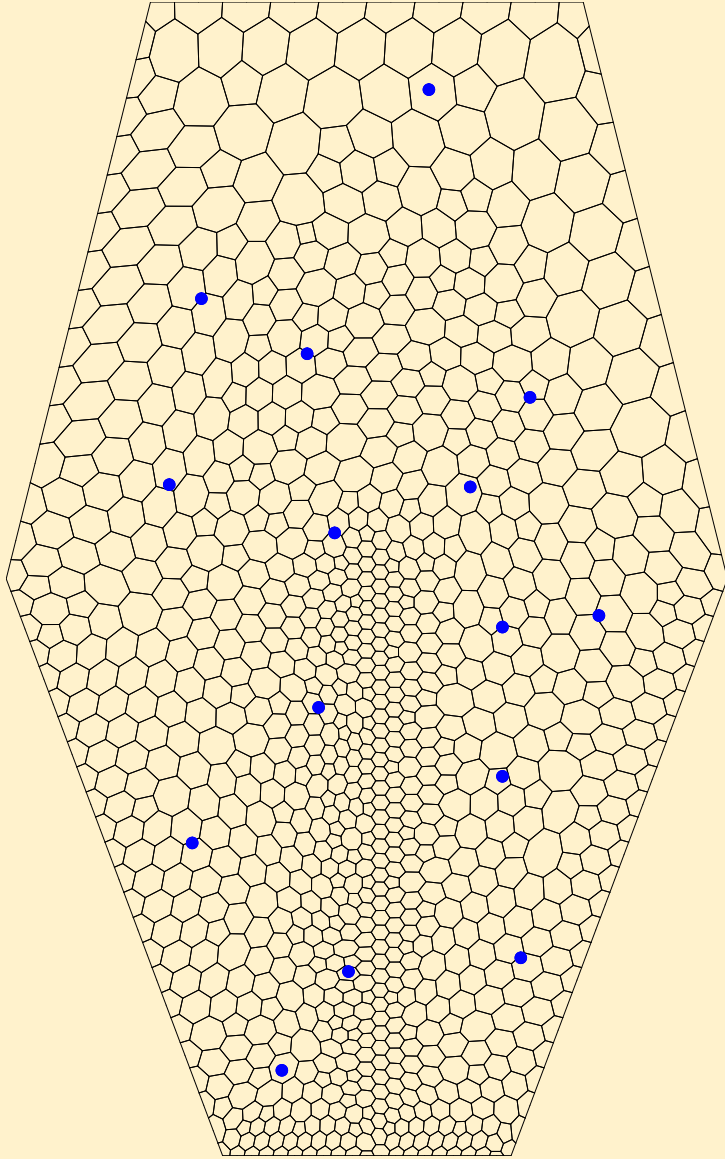
Layer 3



Observations

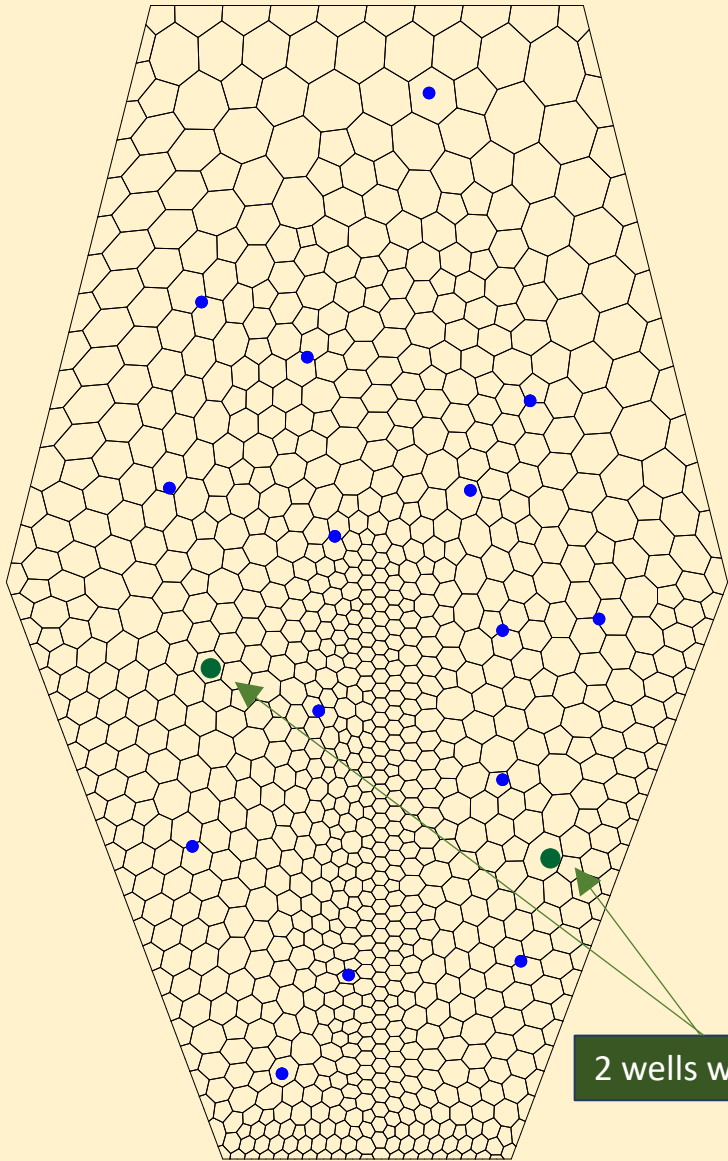
15 observation wells (stress period 1 before any pumping)

- 15 head observations in upper layer (measurement noise stdev = 0.1 m)
- 15 head observations in lower layer (measurement noise stdev = 0.1 m)
- 15 vertical head-difference observations (measurement noise stdev= 0.01m)



“Reality” heads over entire model domain

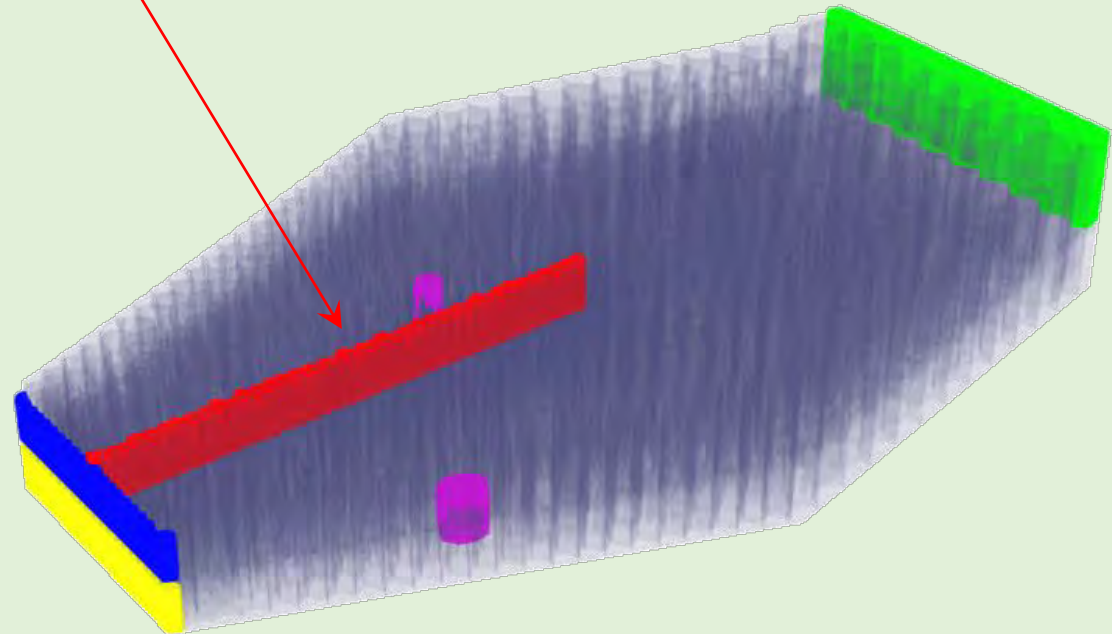
Prediction



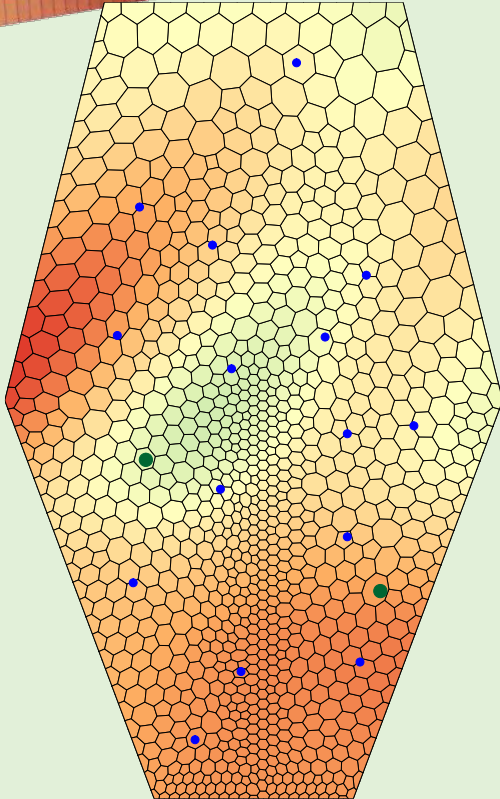
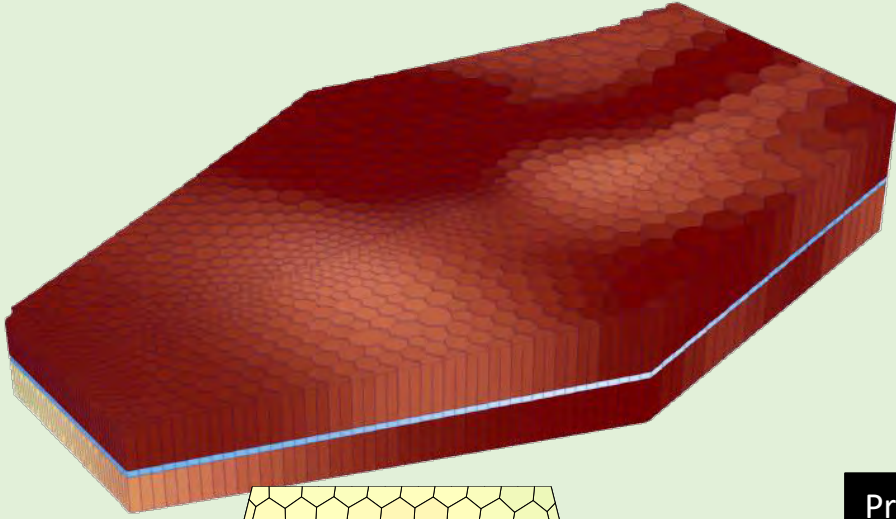
2 wells which pump only in stress period 2

Question

How much will inflow into the drain decrease once both of these bottom layer wells start pumping at 20 m³/day



Data Space Inversion



What is data space inversion?

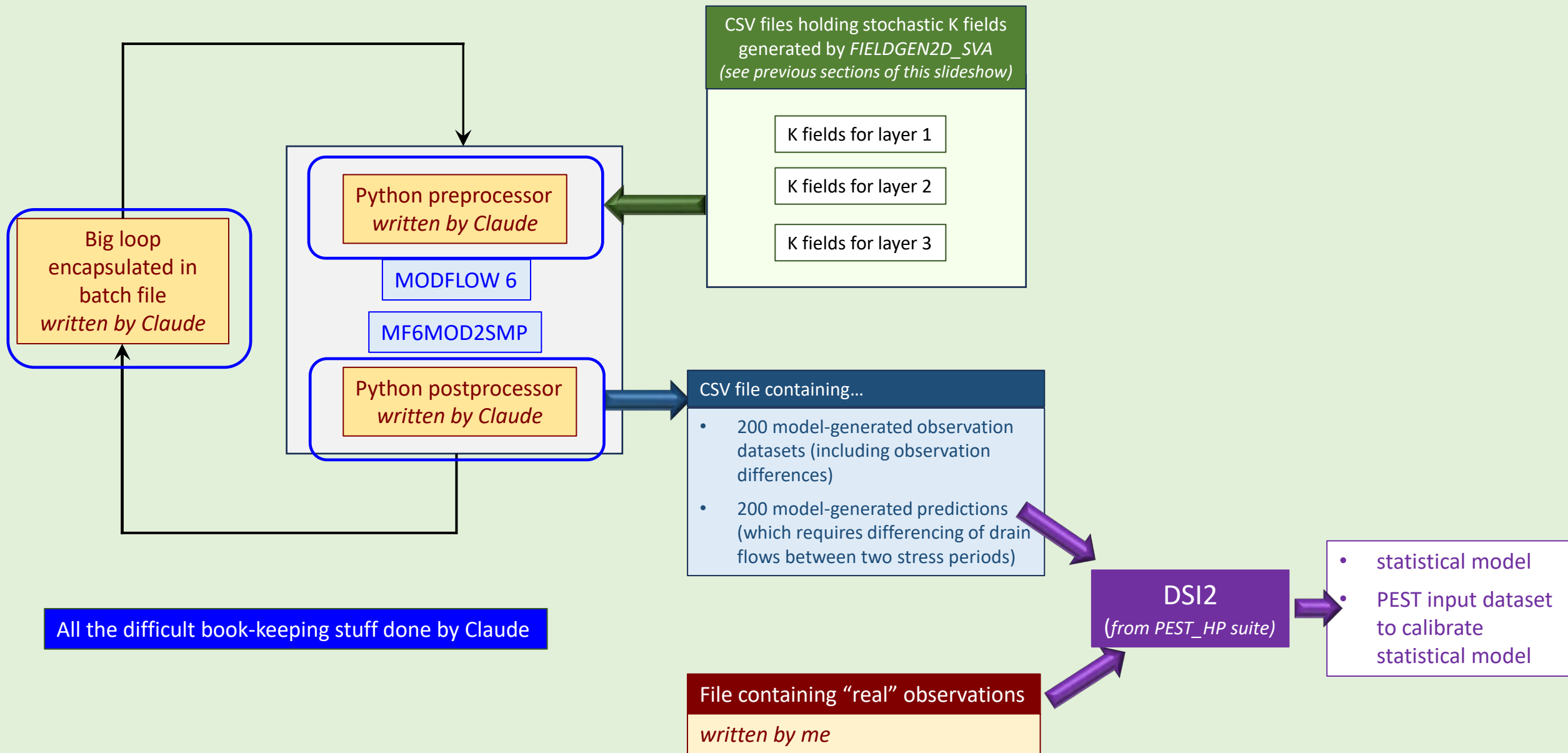
- Run the real model many times using different stochastic hydraulic property fields.
- Calculate observations and predictions during each model run.
- Use these to train a statistical model that links the measured past to the predicted future.
- Calibrate this statistical model, make a prediction and evaluate its uncertainty.

Problems in preparing for implementation of DSI with our particular model

1. How to populate each layer of the model with a stochastic field on each occasion that the model runs (we will use 200 parameter fields to train the DSI statistical model).
2. How to run the model repeatedly.
3. How to read model heads and calculate model inter-layer head differences after each model run, and how to add these to an ever-growing CSV file that records model-generated counterparts to observations and predictions calculated using each stochastic K field.
4. How to calculate the difference between the pre- and post-pumping drain outflow on each occasion that the model runs, and add these to the same every-growing CSV file.

Claude did all of these things

Dataflow



Talking to Claude

I asked Claude

I want to run the model 5 times. (Later it will be many more times than this.) For the first model run, please read the sim1 column of CSV files in the *geostats* folder named *fields_lay1.csv*, *fields_lay2.csv*, *fields_lay3.csv*. The contents of the sim1 column of each of these CSV files must populate files *k1.ref*, *k2.ref* and *k3.ref* respectively (read by MODFLOW 6). During the second model run, the contents of the sim2 column of files *fields_lay1.csv*, *fields_lay2.csv*, *fields_lay3.csv* must populate *k1.ref*, *k2.ref* and *k3.ref* respectively. The model must be run after these input files have been populated in this manner.

I have prepared a file named *model.bat*. You will see that it runs both MODFLOW 6 and a program named MF6MOD2SMP. You should be familiar with this program. See *CLAUDE.md* and your MODFLOW 6 skills and Groundwater Utility skills.

After each model run has taken place, I would like to add a row to a CSV file. This row should contain the contents of two files. The first is *bores_mf6.smp*. This is written by MF6MOD2SMP. I would like the head in each bore (this is in the final column of file *bores_mf6.smp*) recorded. But only those on 02/01/2000 (not on the second day pertaining to the second stress period). The horizontal header of the new CSV file should be "realization,b1u,b2u" etc. Then on each row should be the realization number, followed by the head in each of these bores on 02/01/2000. Then, the final column of this file should reflect the contents of file *model.drn.obs.csv*. This is the output file of the DRN OBS package. (You helped me prepare the input file for this package in a previous session). I would like the final column of the CSV file to contain the drain flow difference between stress periods 1 and 2. This is the difference between the numbers that appear in the final column of *model.drn.obs.csv*.

Please review what I have just written and tell me if I need to make something clearer for you. Then tell me how you would go about this task. For example, you may create a batch file for me that places my existing *model.bat* in a loop. At the moment, just to test things out, I would like to do only 5 model runs. I will do many more model runs later when I equip files *fields_lay*.csv* in the *geostats* folder with many more columns reflecting many more realizations of K fields.

What Claude did

Claude asked me a few questions, gave me a few options (e.g. whether the model should be run repeatedly using a batch loop or Python script). Then did what I asked.

It did not take Claude long to set it all up. It ran perfectly.

(But I checked that outputs of this process were what I expected)

Talking to Claude

Then I asked Claude

That is great. However I would like a slight modification. Can we add to the final CSV file 15 columns. Actually these should be placed before the drain flow difference column. Name these columns "B1U-B1L", "B2U-B2L", etc. The contents of these columns should be the difference between heads in respective bores (upper layer minus lower layer) for the first stress period - i.e. for 02/01/2000.

What Claude did

Thought process

For upper minus lower I need the lower-bore heads too, but the current `bores_mf6.smp` only has the 15 upper bores. Let me check why — the interpolation table it comes from.

Ran a command

The interpolation table `bores.ntb` already contains all 30 bores (B1U-B15U and B1L-B15L), but the current on-disk `bores_mf6.smp` has only the 15 upper ones — it's stale. Let me confirm that a fresh `mf6mod2smp` run emits all 30, by checking the gwutil notes.

Ran a command

Confirmed — `bores.ntb` holds all 30 bores (lowers use layer-3 nodes 2018-3027), and `mf6mod2smp` writes one value per bore in the table. The on-disk `bores_mf6.smp` is just stale (pre-lower-bores); every real run via `model.bat` regenerates it with all 30. So the lower heads will be there at collection time.

Updating `collect_row.py` to add the 15 `BnU-BnL` difference columns before `drn_diff`.

Updated tasks, created a file

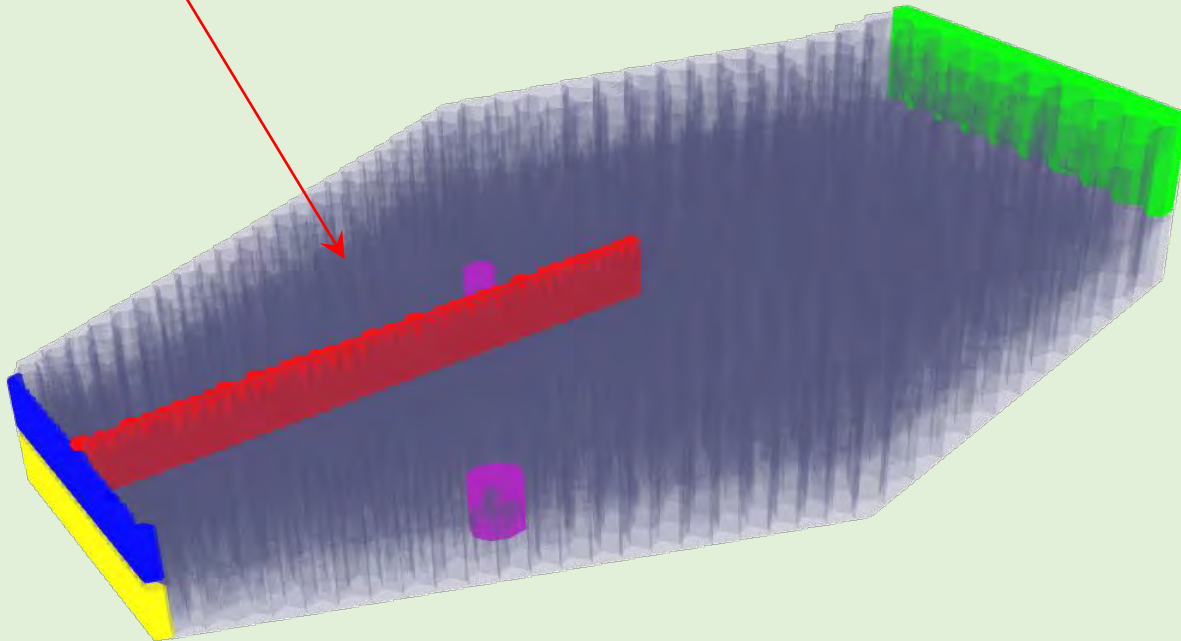
Now testing against a synthetic 30-bore smp (adding lower bores, since the on-disk one is stale/upper-only).

Answers to my question

(from PEST and POSTDSIMOD) after calibration of the DSI statistical model created by DSI2

My original question

How much will inflow into the drain decrease once each of these wells starts pumping at 20 m³/day



Difference in flow to DRN cells induced by pumping in layer 2

True value = 5.98 m³/day

not bad!

Prior mean = 10.01 m³/day

Prediction of DSI statistical model (i.e. posterior mean) = 5.57 m³/day

Uncertainties (calculated using calibrated DSI statistical model)

Prior uncertainty standard deviation of drainflow difference = 7.6 m³/day

Posterior uncertainty standard deviation = 5.84 m³/day

There is some reduction in uncertainty.

But models are inherently better at predicting differences than absolutes. There would be more calibration-induced uncertainty reduction for predictions of actual drain flow rather than drain flow difference.

Nevertheless, the head observations and head difference observations do reduce uncertainty a little. (But this is mitigated by large levels of K heterogeneity, especially in the aquiclude.)

Using Pilot Points for iid Parameterization

Using pilot points

Traditional pilot point usage

- Assign hydraulic property values to pilot points.
- Undertake spatial interpolation to model grid.
- Interpolation to model grid can be anisotropic if appropriate.
- The covariance matrix assigned to pilot point parameters for regularization (if using PEST) or uncertainty analysis (if using PESTPP-IES) should also be anisotropic if interpolation is anisotropic.

Alternative pilot point usage

- Assign iid values to pilot points.
- Calculate hydraulic property values for cells of model grid using spatial averaging over many pilot points.
- These are the same spatial averaging functions as those used by FIELDGEN2D_SVA (see earlier in this slideshow). They can easily embody spatially varying anisotropy (and other spatially variable geostatistical hyperparameters).
- Because pilot point parameters are iid's, their covariance matrix is the identity matrix.

BTW

An “iid” variable is an independent and identically distributed random variable, normally possessing a standard normal distribution.

This does not change even if the spatial averaging functions changes.

Hence PEST/PEST++ can adjust conceptual point geostatistical hyperparameters that govern the shapes of hydraulic property heterogeneity at the same time as it adjusts pilot point iid parameters (which determine the locations of heterogeneity). Meanwhile the iid parameters retain their original covariance matrix.

This is known as “hierarchical inversion”.

Using pilot points

I asked Claude

I want to distribute about 300 points throughout the footprint of the model domain. I would like to apply the following specifications.

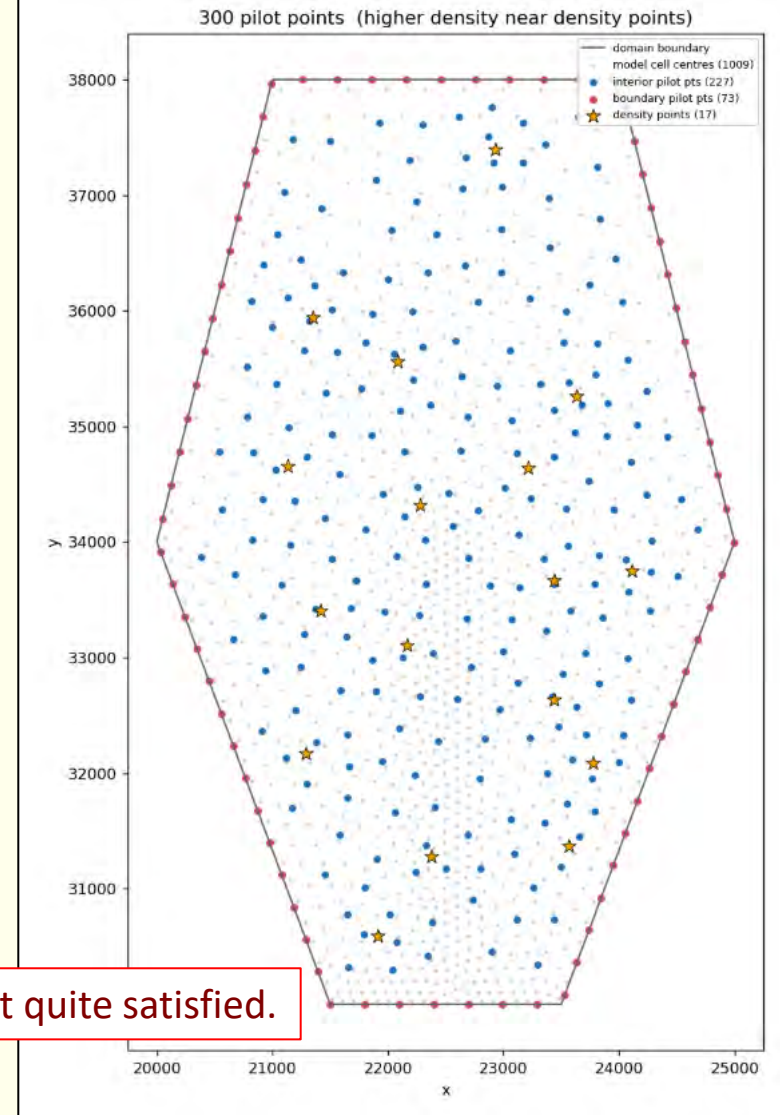
Slightly higher spatial density in the vicinity of points whose locations are specified in the second and third columns of file *density_points.dat*. (These are the locations of observation and pumping wells.)

Outer pilot points should be close to the boundary of the model domain. (Eventually I am going to be interpolating from these points to cells of the model domain; I want to avoid extrapolation.)

No model cell centre should be further from at least two pilot points than 500 metres. (If we need more than 300 pilot points to achieve these objectives, that is ok.)

Record the pilot point x and y coordinates in the second and third columns of a file named *pp.dat*. The first column of this file should be pilot point names. Names are pp1, pp2 etc

Claude wrote the pilot points file and gave me a picture



But I wasn't quite satisfied.

Using pilot points

So I asked Claude

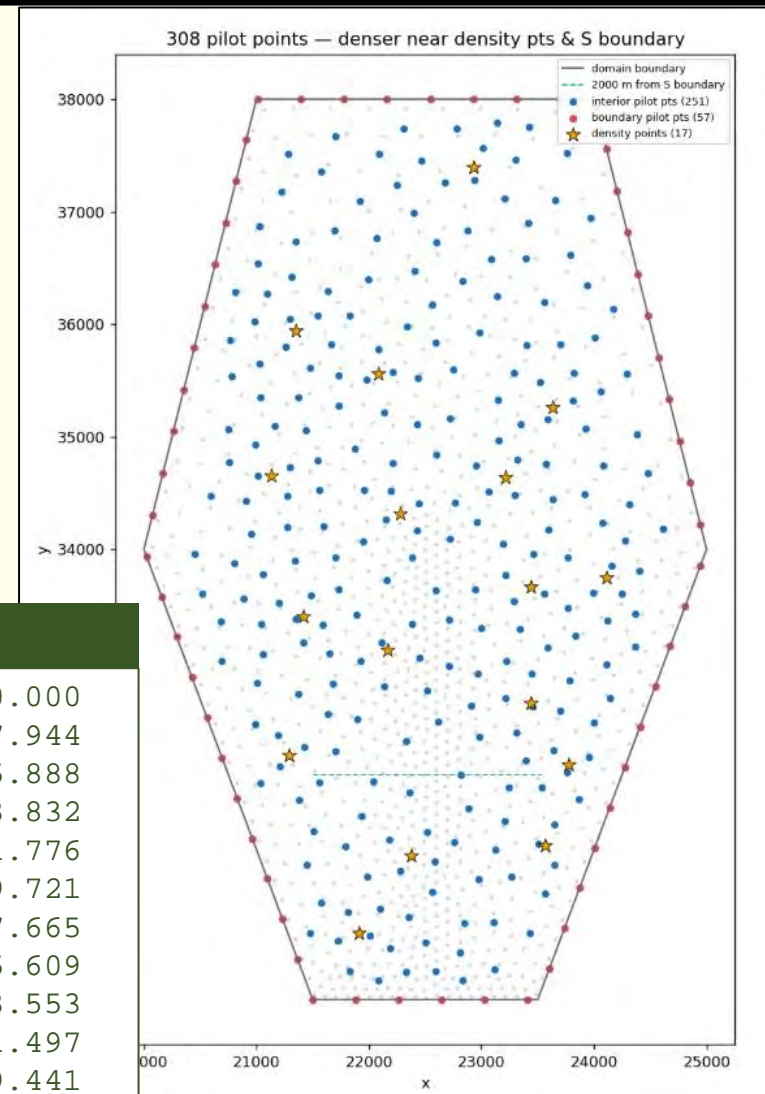
That is good.

However let me ask for a small alteration to pilot points emplacement philosophy. The boundary points do not need to be exactly on the boundary. They can be a little outside or a little inside the boundary. And I don't care if they are a little more sparse on the boundary if this allows more pilot points in other places (as long as the rules specified in the previous prompt are obeyed).

Also, I'd like slightly greater pilot point density within 2000 metres of the southern boundary.

You can increase pilot points to 310 if this gives you more room to move. These are just small alterations. What you have done is really good.

Claude wrote another pilot points file and gave me a picture



File *pp.dat*

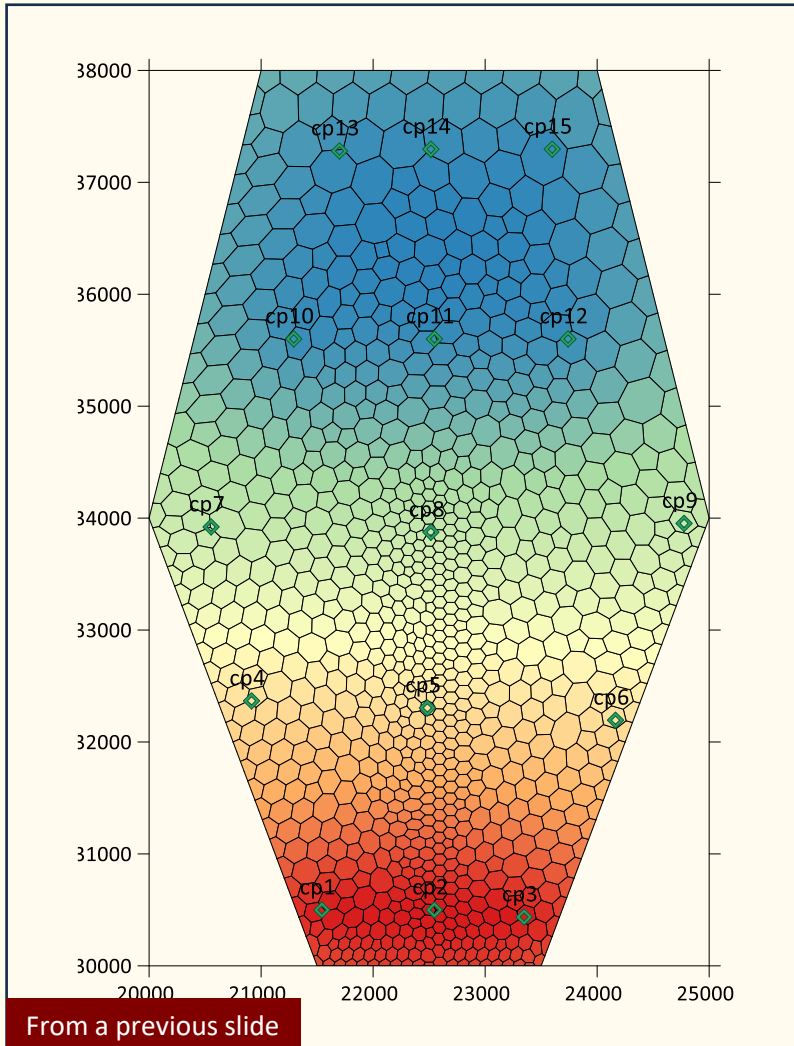
pp1	21500.000	30000.000
pp2	21365.771	30357.944
pp3	21231.542	30715.888
pp4	21097.313	31073.832
pp5	20963.084	31431.776
pp6	20828.855	31789.721
pp7	20694.626	32147.665
pp8	20560.397	32505.609
pp9	20426.168	32863.553
pp10	20291.939	33221.497
pp11	20157.710	33579.441
pp12	20023.481	33937.385
etc		

Pilot points and spatial averaging

Each cell in each layer of the model grid already has geostatistical hyperparameters associated with it.

These were interpolated from conceptual points by PLPROC prior to using FIELDGEN2D_SVA.

See a previous section of this slideshow.



The plan

Three set of iids will be ascribed to the pilot points – one set for each model layer.

PLPROC function *spatial_average_2d()* will implement spatial averaging (three times – once for each layer) in order to assign a K value to every cell of each model layer.

Spatially averaged properties will then be recorded in files named *k1.ref*, *k2.ref* and *k3.ref* that are accessible by the NPF package of MODFLOW 6 through its OPEN/CLOSE functionality.

Pilot points and spatial averaging

I asked Claude

Please modify *pp.dat* in the following ways:
Add three columns to this file. Equip each of these columns with random numbers sampled from a standard normal distribution (mean zero, standard deviation of 1.0). Add headers to all columns. These headers should be *pp_id*, *x*, *y*, *k1*, *k2*, *k3*.

Claude gave me the following file...

<i>pp_id</i>	<i>x</i>	<i>y</i>	<i>k1</i>	<i>k2</i>	<i>k3</i>
pp1	21500.000	30000.000	0.148865	-0.338755	-0.338017
pp2	21365.771	30357.944	0.027654	-2.048296	0.740515
pp3	21231.542	30715.888	0.257651	-0.008767	0.110246
pp4	21097.313	31073.832	-0.582966	1.208681	0.922514
pp5	20963.084	31431.776	1.376575	-0.420686	-1.057042
pp6	20828.855	31789.721	-0.498702	0.937862	0.657901
pp7	20694.626	32147.665	0.350256	-0.015484	0.896740
pp8	20560.397	32505.609	0.881718	-0.699740	0.381857
pp9	20426.168	32863.553	1.157156	-1.884849	-0.415027
pp10	20291.939	33221.497	0.930647	1.744777	-1.311294
pp11	20157.710	33579.441	0.561558	0.332837	-2.267849
pp12	20023.481	33937.385	0.009891	-0.551470	0.308277
pp13	20076.499	34305.994	0.540659	-0.748099	0.430443
pp14	20169.216	34676.865	0.235326	2.388366	0.096370
pp15	20261.934	35047.735	-0.318179	-1.953446	-0.812484
pp16	20354.651	35418.606	-0.074046	-0.803991	-0.507316
etc					

Pilot points and spatial averaging

I asked Claude

I would like to use PLPROC. I want it to take iid values that are contained in file *pp.dat* (in columns headed k1, k2, and k3) to populate every cell of the DISV grid. Values in the k1 column pertain to layer 1. Values in the k2 column pertain to layer 2. Values in the k3 column pertain to layer 3.

I would like to use the PLPROC function named *spatial_average_2d()*. For layers 1, 2 and 3 the hyperparameter values that are needed for PLIST arguments of this function can be found in files *avgfunc_lay1.dat*, *avgfunc_lay2.dat* and *avgfunc_lay3.dat*. See the headers to these columns for what they mean (you can relate them to PLPROC function arguments). PLPROC must read these files to obtain hyperparameters for every cell in the model grid.

The coordinates of voronoi grid cell centres can be found in file *disv_cellcent.dat*.

I would like K values written to files *k1.ref*, *k2.ref* and *k3.ref* that are expected by MODFLOW 6.

Are you able to do this? Ask any questions that you need. The PLPROC manual is in the docs folder.

Written during work documented in previous sections of this slideshow. These were used by FIELDGEN2D_SVA.

Claude said (among many other things)

What it does, per your spec:

- Reads the 308 pilot points from *pp.dat* (k1→layer 1, k2→layer 2, k3→layer 3).
- Target CLIST = the 1009 cell centres from *disv_cellcent.dat*.
- *spatial_average_2d()* PLIST arguments (mean, variance←sill, a←aa, anis, bearing) read from *avgfunc_lay1/2/3.dat* via *read_column_data_file* (row order = node order, so they land on the matching cells).
- Kernel = **exponential**, transform='log' → $K = \text{mean} \cdot 10^{(\text{convolution})}$. The pilot area PLIST is a constant (it cancels exactly in the normalized convolution).
- Writes one K value per line to each *kN.ref* — the free-format array MODFLOW 6 NPF reads via OPEN/CLOSE.

Oops!! I forgot to tell Claude how to associate an area with each pilot point. (This is required by the *spatial_average_2d()* function of PLPROC.)

Pilot points and spatial averaging

I asked Claude

You are correct that the area associated with each pilot point is relative. But they should not be the same because pilot point spatial density differs in different parts of the model domain. So PLPROC needs to read a file that lists the area that is associated with each pilot point. I would like you to write this file and name it *pp_area.dat*.

To calculate the relative area associated with each pilot point, would you please calculate the average distance between each pilot point and its three closest neighbours. Then square this number. These numbers can be recorded in file *pp_area.dat*. Then modify the PLPROC script that you previously wrote to read this area.

Thanks Claude

Claude said (among many other things)

Thought process

Makes sense — non-constant areas weight each pilot point by its footprint. Let me compute $\text{area} = (\text{mean distance to 3 nearest pilot neighbours})^2$, write *pp_area.dat*, then wire it into the script.

Updated tasks, ran a command

Areas span 225-436 m (mean 3-nn distance), i.e. $\sim 50,600\text{--}189,900\text{ m}^2$ — nicely tracking the local pilot spacing. Now wiring *pp_area.dat* into the script.

Edited a file

Now re-checking the field with variable areas, then updating CLAUDE.md and the skill.

Ran a command

Field stays positive and correctly scaled with the variable areas. Updating CLAUDE.md and the skill to reflect *pp_area.dat*.

Pilot points and spatial averaging

PLPROC script (written by Claude)

```
# =====
# make_kref_pp.plp
#
# Populate every cell of the 3-layer DISV voronoi grid with a hydraulic-
# conductivity value by spatially averaging the pilot-point normal deviates in
# pp.dat onto the grid with the PLPROC function spatial_average_2d().
#
# SOURCE points : 308 pilot points in pp.dat
#   col1 pp_id (char) col2 x col3 y col4 k1 col5 k2 col6 k3
#   k1/k2/k3 are independent standard-normal deviates (one set per layer).
#
# TARGET points : 1009 voronoi cell centres in disv_cellcent.dat
#   col1 cell (int) col2 x col3 y
#
# AVERAGING FN : per-cell hyperparameters in avgfunc_lay{1,2,3}.dat
#   col1 node col2 mean col3 sill col4 aa col5 anis col6 bearing
#   sill -> variance_plist , aa -> a_plist , anis -> anis_plist ,
#   bearing -> bearing_plist , mean -> mean_plist .
#
# KERNEL      : exponential      f(r) = exp(-r/a)
# TRANSFORM   : log -> K = mean * 10^(convolution)
#              positive, log-normal; local median = mean, local log10 spread
#              = sqrt(sill). (mean per layer ~ 2-4 / 5e-6 / 1-1.5 = K medians.)
#
# OUTPUT      : k1.ref, k2.ref, k3.ref (one K value per line, free format),
#              read by MODFLOW 6 NPF via OPEN/CLOSE, one file per layer.
#
# Run on Windows in the voro folder:  plproc make_kref_pp.plp
# =====

#-- 1. TARGET CLIST : 1009 voronoi cell centres -----
cl_grid = read_list_file(file='disv_cellcent.dat', &
                        dimensions=2, &
                        id_type='integer', &
                        skiplines=1)

#-- 2. SOURCE CLIST : pilot points + their per-layer normal deviates -----
#   pp.dat cols: 1 pp_id | 2 x | 3 y | 4 k1 | 5 k2 | 6 k3
cl_pp = read_list_file(file='pp.dat', &
                      dimensions=2, &
                      id_type='character', &
                      skiplines=1, &
                      plist='k1';column=4, &
                      plist='k2';column=5, &
                      plist='k3';column=6)
```

```
# Relative area (footprint) of each pilot point, read from pp_area.dat.
# area = (mean distance to the point's 3 nearest pilot-point neighbours)^2.
# Row order of pp_area.dat matches pp.dat, so read_column_data_file transfers
# by position onto cl_pp. In spatial_average_2d() the area weights each source
# point's contribution to the convolution: points in sparse regions (larger
# footprint) carry proportionally more weight. (A CONSTANT area would cancel
# exactly; these deliberately vary, so they matter. Only relative values count.)
# pp_area.dat cols: 1 pp_id | 2 area
area = new_plist(reference_clist=cl_pp, value=0.0)
read_column_data_file(file='pp_area.dat', skiplines=1, &
                    plist=area;column=2)

# =====
# LAYER 1 -> k1.ref (uses pp.dat column k1 and avgfunc_lay1.dat)
# =====
mean1 = new_plist(reference_clist=cl_grid, value=0.0)
var1 = new_plist(reference_clist=cl_grid, value=0.0)
aal = new_plist(reference_clist=cl_grid, value=0.0)
anis1 = new_plist(reference_clist=cl_grid, value=0.0)
bear1 = new_plist(reference_clist=cl_grid, value=0.0)

read_column_data_file(file='avgfunc_lay1.dat', skiplines=1, &
                    plist=mean1;column=2, &
                    plist=var1;column=3, &
                    plist=aal;column=4, &
                    plist=anis1;column=5, &
                    plist=bear1;column=6)

klgrid = new_plist(reference_clist=cl_grid, value=0.0)
klgrid = kl.spatial_average_2d(area_plist=area, &
                              mean_plist=mean1, &
                              variance_plist=var1, &
                              a_plist=aal, &
                              anis_plist=anis1, &
                              bearing_plist=bear1, &
                              avefunctype=exponential, &
                              transform='log')

write_column_data_file(file='k1.ref', header='no', plist=klgrid)
```

Pilot points and spatial averaging

```
# =====  
# LAYER 2 -> k2.ref (uses pp.dat column k2 and avgfunc_lay2.dat)  
# =====  
mean2 = new_plist(reference_clist=cl_grid, value=0.0)  
var2 = new_plist(reference_clist=cl_grid, value=0.0)  
aa2 = new_plist(reference_clist=cl_grid, value=0.0)  
anis2 = new_plist(reference_clist=cl_grid, value=0.0)  
bear2 = new_plist(reference_clist=cl_grid, value=0.0)  
  
read_column_data_file(file='avgfunc_lay2.dat', skiplines=1, &  
                      plist=mean2;column=2, &  
                      plist=var2;column=3, &  
                      plist=aa2;column=4, &  
                      plist=anis2;column=5, &  
                      plist=bear2;column=6)  
  
k2grid = new_plist(reference_clist=cl_grid, value=0.0)  
k2grid = k2.spatial_average_2d(area_plist=area, &  
                              mean_plist=mean2, &  
                              variance_plist=var2, &  
                              a_plist=aa2, &  
                              anis_plist=anis2, &  
                              bearing_plist=bear2, &  
                              avefunctype=exponential, &  
                              transform='log')  
  
write_column_data_file(file='k2.ref', header='no', plist=k2grid)
```

```
# =====  
# LAYER 3 -> k3.ref (uses pp.dat column k3 and avgfunc_lay3.dat)  
# =====  
mean3 = new_plist(reference_clist=cl_grid, value=0.0)  
var3 = new_plist(reference_clist=cl_grid, value=0.0)  
aa3 = new_plist(reference_clist=cl_grid, value=0.0)  
anis3 = new_plist(reference_clist=cl_grid, value=0.0)  
bear3 = new_plist(reference_clist=cl_grid, value=0.0)  
  
read_column_data_file(file='avgfunc_lay3.dat', skiplines=1, &  
                      plist=mean3;column=2, &  
                      plist=var3;column=3, &  
                      plist=aa3;column=4, &  
                      plist=anis3;column=5, &  
                      plist=bear3;column=6)  
  
k3grid = new_plist(reference_clist=cl_grid, value=0.0)  
k3grid = k3.spatial_average_2d(area_plist=area, &  
                              mean_plist=mean3, &  
                              variance_plist=var3, &  
                              a_plist=aa3, &  
                              anis_plist=anis3, &  
                              bearing_plist=bear3, &  
                              avefunctype=exponential, &  
                              transform='log')  
  
write_column_data_file(file='k3.ref', header='no', plist=k3grid)  
  
stop
```

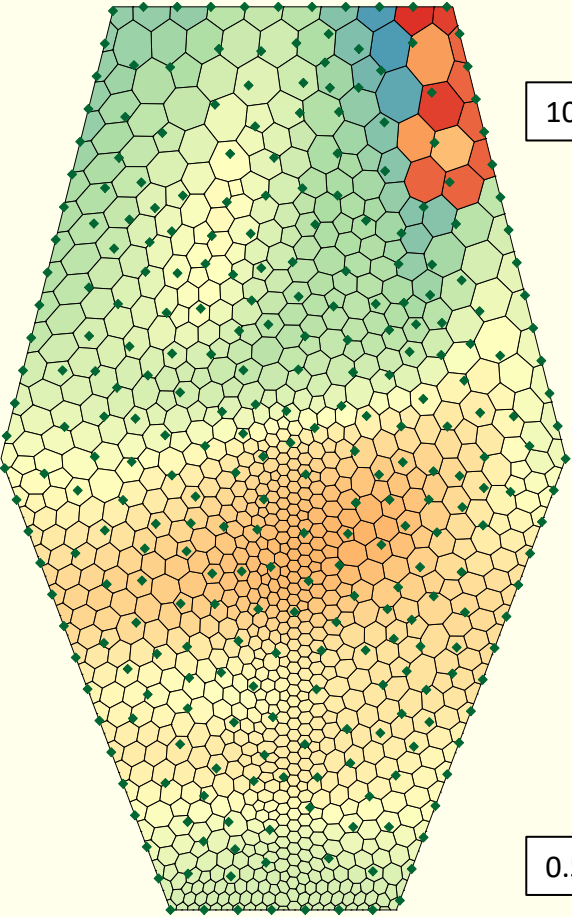
PLPROC ran this script without any problems.

Pilot points and spatial averaging

Layer 1

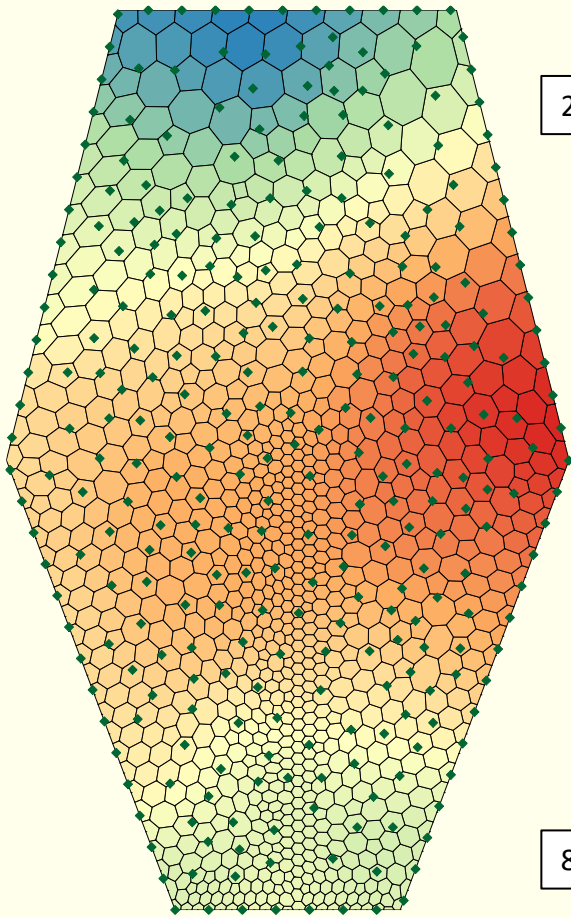
Layer 2

Layer 3



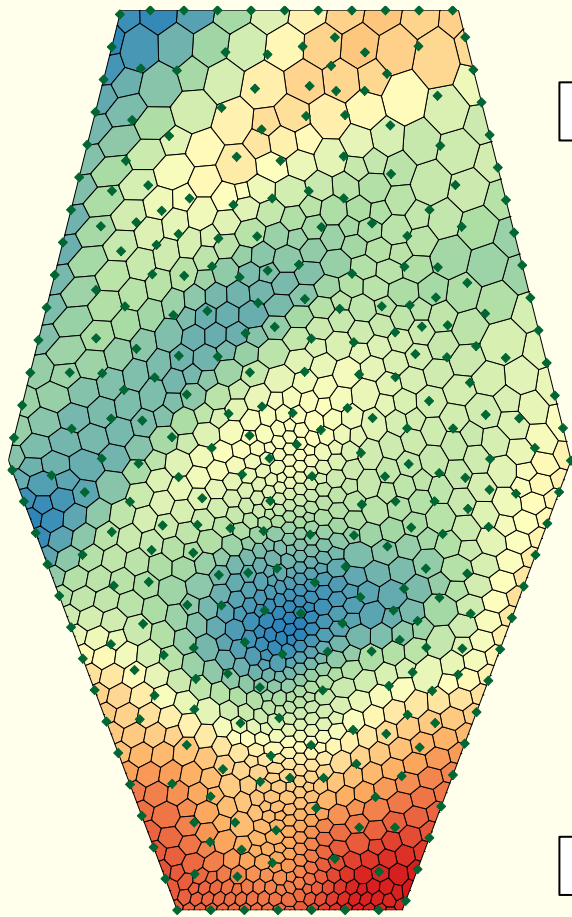
100 m/day

0.5 m/day



2.0E-5 m/day

8.0E-7 m/day



1.5 m/day

0.3 m/day

K fields derived from random iid values supplied to pilot points (by PLPROC)

Preparing for a PEST Run

Setting up for a PEST run

I asked Claude

I want to calibrate the model that we have been working on using PEST. I have placed the PEST manual *pestman1.pdf* in the *docs* subfolder. Please read that manual, but don't do anything until I ask you to.

I said to Claude

First let's look at the model that PEST will run. I have prepared a batch file named *model.bat*. This runs PLPROC (to calculate model cell K values from pilot point values), then MODFLOW 6, and then MF6MOD2SMP to calculate heads at the sites of observation bores. Spatially-interpolated heads are calculated for the first and second stress periods. However I only want to use heads from the first stress period. I also want to use vertical head differences between bores which are at the same horizontal locations in the upper and lower layers. You can recognize upper layer bores by the "u" that ends their names, and lower layer bores by the "l" that ends their names. Bores at the same location have the same prefix.

Can you write a program that can calculate these head differences on each occasion that the model is run?

What is the best thing to do? Have two instruction files – one that reads heads (for only the first stress period) from the MF6MOD2SMP output file, and another that reads head differences from the differencing output file? Or should your program record first stress period heads and head differences in a single file, so that only a single instruction file is required?

Claude said (among many other things)

Which version of PEST would you like to use? PEST? PEST_HP? PEST++?

Which parameters should PEST adjust? Pilot-point parameters for K? Conductance parameters?

Which observations should comprise calibration targets? Heads? Drain flow? Both?

Claude said (among many other things)

The second option is the clean way to do it.

What are the observed values of head and head differences to record in the PEST control file?

Setting up for a PEST run

I said to Claude

Observed heads, head differences and weights reside in a file named *observations.dat*. Please build these into the PEST control file. Note that heads and head differences are assigned to different observation groups.

K parameters are iids associated with pilot points. These are listed in file *pp.dat* (read by PLPROC) in columns labelled k1, k2 and k3. These iid parameters must be declared as untransformed (rather than log-transformed) in the PEST control file. Lower and upper bounds must be minus 10 and 10 respectively.

When you write the PEST control file, please do not add Tikhonov regularization. I will look after this myself.

I said to Claude

Can you add a “singular value decomposition” section to that PEST control file. Also, in the “parameter data” section, can you re-order the parameters so that all the layer 1 pilot point parameters are listed before all the layer 2 pilot point parameters etc.

Claude did the following...

- Created a template file of *pp.dat* named *pp.dat.tpl*.
- Wrote a Python program to do vertical differencing of borehole heads on each occasion that the model is run.
- The program records these, as well as stress period 1 heads, in a single output file.
- Added this program to the model batch file.
- Wrote an instruction file to read this output file.
- Wrote a PEST control file.

Claude said

All done. I'll leave it to you to check my work with PESTCHEK, TEMPCHEK and INSCHEK.

I've set NOPTMAX to zero in the PEST control file so that you can use PEST to do one model run to make sure that everything works ok.

Setting up for a PEST run

When I ran PEST...

```
E:> pest model

PEST Version 18.27. Watermark Numerical Computing.

PEST is running in parameter estimation mode.

PEST run record: case model
(See file model.rec for full details.)

Model command line:
model.bat

Running model .....

Running model 1 time...
E:\myclaude\modflow6\MODFLOW6\voro>del k1.ref
Could Not Find E:\myclaude\modflow6\MODFLOW6\voro\k1.ref
```

PLPROC was not reading the input file containing the values of iid parameters (i.e. file *pp.dat*).

PEST was supposed to write this file using the template file *pp.dat.tpl* that Claude had prepared for it.

Files written by Claude

- *pp.dat.tpl*
- *model_obs.ins*
- *case.pst*
- Python script *make_model_obs.py*

Claude also modified the model batch file *model.bat*

Then I recalled what PESTCHEK had said (which I had previously ignored)...

Setting up for a PEST run

When I ran PESTCHEK:

```
E:> pestchek model.pst
```

```
PESTCHEK Version 18.27. Watermark Numerical Computing.
```

```
Errors ----->
```

```
No errors encountered.
```

```
Warnings ----->
```

```
NOPTMAX provided as zero. No optimisation iterations will be carried out;  
objective function and residuals will be recorded for initial parameter  
estimates only.
```

```
No SVDMODE or LSQRMODE variables appear in PEST control model.pst. The user  
should be aware of how powerful these can be. Try one of these (especially  
SVD) if PEST does not perform as well as you think it should.
```

```
Where many parameters are being estimated as in the present case,  
significant memory can be saved by setting all of ICOV, ICOR and IEIG to  
zero in the "control data" section of the PEST control file.
```

```
File model.pst appears to be missing carriage returns and may not be read  
properly by PEST or some of its utilities. Has it been imported from a  
UNIX machine? Try running a UNIX2DOS converter, or reading it into EDIT,  
making a change and re-writing it.
```

Claude works in its own linux sandbox on my WINDOWS machine when preparing my files. I told Claude about this PESTCHEK error and it added CRLF (carriage-return line-feed) to all PEST-pertinent files that it wrote for me (i.e. the template file, the instruction files and the PEST control file).

Setting up for a PEST run

I calibrated the model using ENSI (which I set up myself) based on the PEST input dataset that Claude had prepared for me

- number of pilot point parameters (3×308) = 924
- ENSI used 60 parameters (20 realizations for each of the k1, k2 and k3 iid parameter sets)

Prediction made by calibrated model

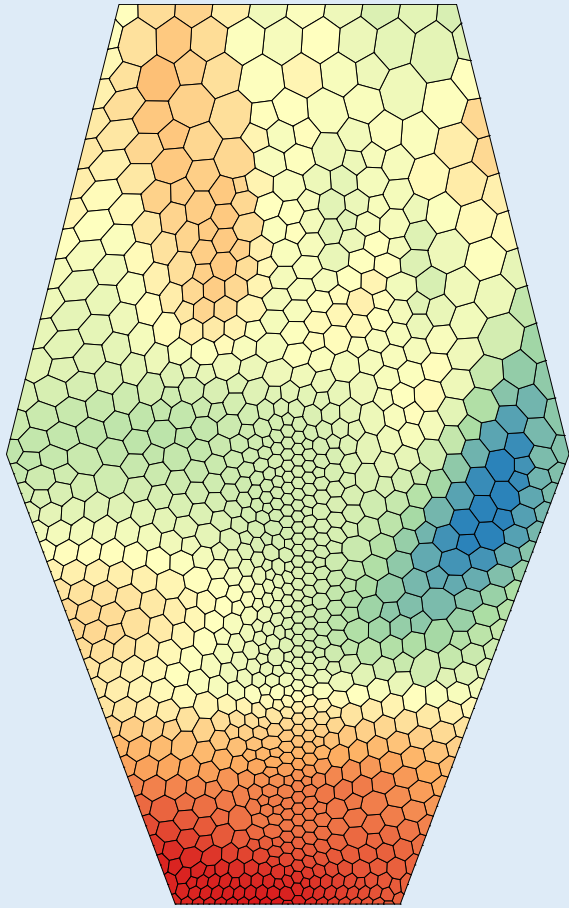
“Real” drain flow difference (see previous section of this slideshow about on DSI) = 5.98 m³/day

Drain flow difference predicted by calibrated model = 6.27 m³/day

Not bad!

Calibrated K fields

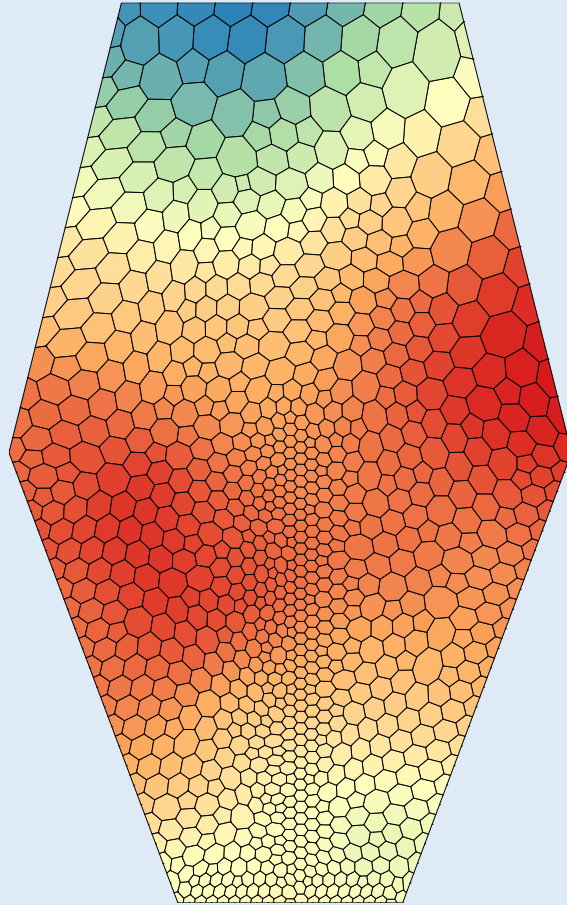
Layer 1



100 m/day

0.5 m/day

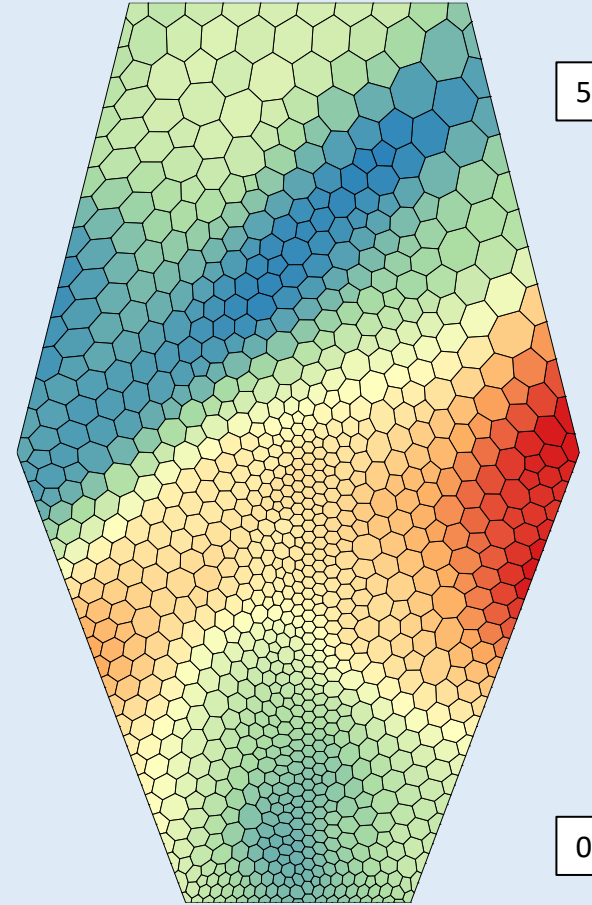
Layer 2



1.08E-5 m/day

7.0E-7 m/day

Layer 3



5.0 m/day

0.3 m/day

Calibrated using ENSI: 20 realizations in each of 3 layers
(These fields would probably have showed less variability if using more realizations.)

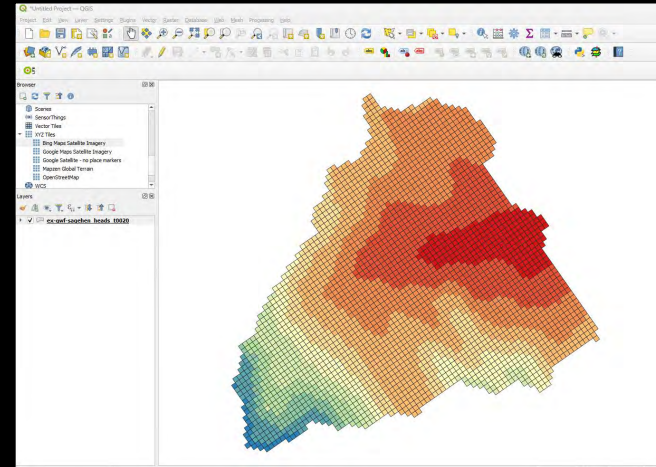
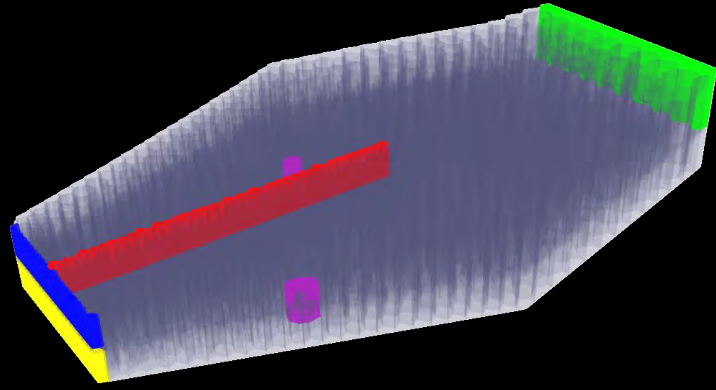
Conclusions

Conclusions

Using AI is NO SUBSTITUTE for knowing what you are doing.

If you know your destination, AI can straighten your path.
But AI cannot provide you with a destination. It can easily take you nowhere.





The End

