DHM Diffusion Hydrodynamic Model

USGS Water Resources Division Special Report 87-4137

Modernizing the Diffusion Hydrodynamic Model

Nicholas Wimer and Ted Hromadka

October 22, 2021

2021 AUTS (ARL-USMA Technical Symposium)

Overview of DHM

- The **Diffusion Hydrodynamic Model (DHM)** is a “Legacy” hydrodynamics code for unsteady flow problems such as rainfall runoff modeling, channel floodplain interface modeling, and other free surface flow problems.

- Developed in the late 1970s to early 1980s (published in 1987)

- Written in Fortran 77 convention

- Repeatedly validated over the years with numerous publications

- Equation formulation and solution method makes for a lightweight, reliable solution to many free surface flow problems

- However, DHM is subject to the limitations of its time, so setting up a new problem is an entirely manual process

GOAL: Leverage advancements in modern programming languages to automate problem creation and data analysis via preprocessing and postprocessing Python scripts

Diffusion Hydrodynamic Model

- Governing flow equations for DHM are derived based on continuity and momentum

q_i = flow rate per unit width in i direction
H = water surface elevation
h = flow depth
S = Friction slope

Continuity:

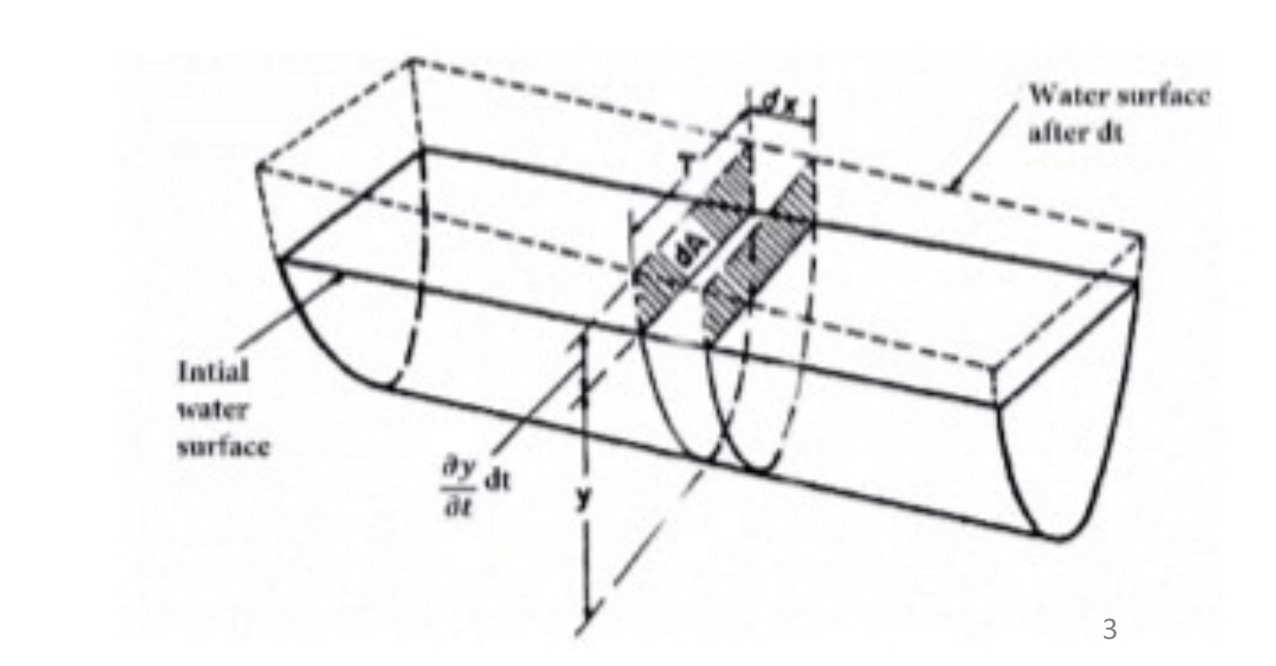
$$\frac{\partial q_i}{\partial x_i} + \frac{\partial H}{\partial t} = 0$$

Momentum:

$$\frac{\partial q_i}{\partial t} + \frac{\partial}{\partial x_i}\left(\frac{q_i^2}{h}\right) + \frac{\partial}{\partial x_j}\left(\frac{q_i q_j}{h}\right) + gh\left(S_i + \frac{\partial H}{\partial x_i}\right) = 0$$

- The solution can be greatly simplified by assuming that diffusion is dominant

$$\frac{1}{gh}\left(\frac{\partial q_i}{\partial t} + \frac{\partial}{\partial x_i}\left(\frac{q_i^2}{h}\right) + \frac{\partial}{\partial x_j}\left(\frac{q_i q_j}{h}\right)\right) \ll S_i + \frac{\partial H}{\partial x_i}$$



Diffusion Hydrodynamic Model

- Once we assume that diffusion is dominate, the equation of motion reduces to

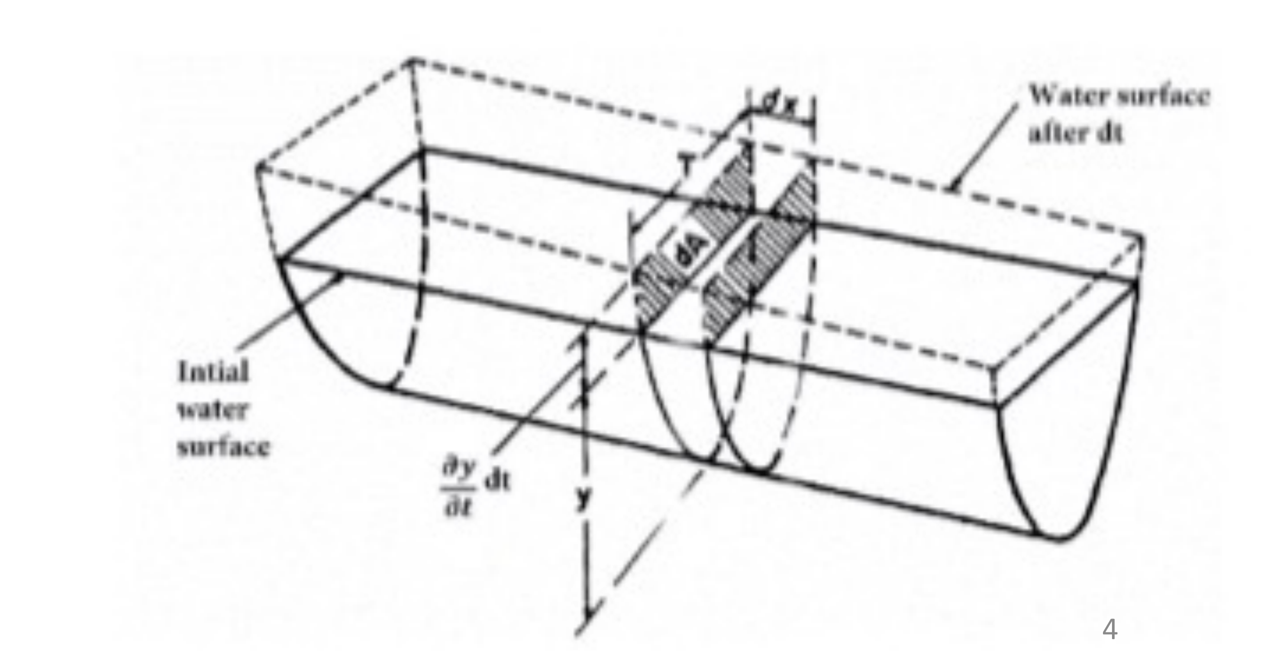
$$q_i = -K_i \frac{\partial H}{\partial x_i}$$

- We can now substitute the above equation for discharge into the continuity equation to form a single equation of motion for DHM

$$\frac{\partial H}{\partial t} = \frac{\partial}{\partial x_i} K_i \frac{\partial H}{\partial x_i}$$

- Where K is a variable derived from the Manning formula

- Due to this simple assumption, we have reduced a coupled system of PDEs into a single PDE

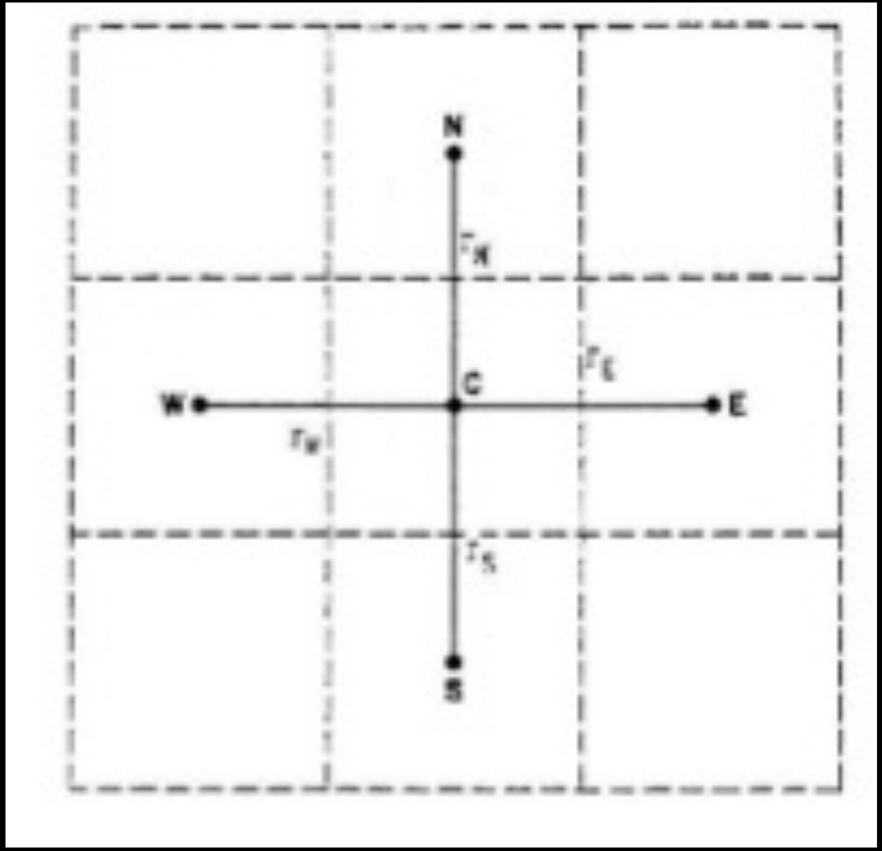


Overview of the Solution Method

- Requires a uniform, structured, square grid
- At each node, Manning's n, elevation, and initial flow depth are required
- Additionally, we must identify each cell's neighbors (N, E, S, W)

- Once set, the solution method proceeds is as follows:

1. Compute average Manning n and geometric quantities for between nodal points
2. Estimate the nodal water surface elevation (H) for next time step $(t + \Delta t)$
3. Estimate the value of m_i (set to 0 for full DHM)
4. Recalculate K_i using the approximate m_i
5. Determine new H at $(t + \Delta t)$
6. Return to step #3 and iterate until K_i matches the mid time step estimates



Full DHM assumption leads to fast convergence!!

DHM Input File

- DHM is run based on the inputs of a single formatted input file
- Two main blocks of the input file:
- The first contains the grid connections (N, S, E, W), manning n value, elevation, and initial water depth for **every cell** in the computational domain
- The second contains the indices of all cells containing the **water channel** and their associated manning n value, width of the channel, depth of the channel, and initial water depth

GOAL: Automate the creation of the DHM input file using files generated by standard GIS tools (shape files and terrain rasters)

METHOD: Use Python and leverage its many packages, mainly Pandas, GeoPandas, and RasterIO

```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

  2   6   0   0  0.08   101.0  0.0

  3   7   1   0  0.08   101.5  0.0

  4   8   2   0  0.08   102.0  0.0

  5   9   3   0  0.08   102.5  0.0

  0  10   4   0  0.08   103.0  0.0

  7  11   0   1  0.08   100.0  0.0

  8  12   6   2  0.08   100.5  0.0

  9  13   7   3  0.08   101.0  0.0

 10  14   8   4  0.08   101.5  0.0

  0  15   9   5  0.08   102.0  0.0

 12  16   0   6  -0.08   100.0  0.0

 13  17  11   7  -0.08   100.5  0.0

 14  18  12   8  -0.08   101.0  0.0

 15  19  13   9  -0.08   101.5  0.0

  0  20  14  10  -0.08   102.0  0.0

 17  21   0  11  0.08   100.0  0.0

 18  22  16  12  0.08   100.5  0.0

 19  23  17  13  0.08   101.0  0.0

 20  24  18  14  0.08   101.5  0.0

  0  25  19  15  0.08   102.0  0.0

 22   0   0  16  0.08   100.0  0.0

 23   0  21  17  0.08   100.5  0.0

 24   0  22  18  0.08   101.0  0.0

 25   0  23  19  0.08   101.5  0.0

  0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

 15  0.015 10.0  6.0 0.0

 14  0.015 10.0  6.0 0.0

 13  0.015 10.0  6.0 0.0

 12  0.015 10.0  6.0 0.0

 11  0.015 10.0  6.0 0.0

 [illegible]

 15 0 0 1 100 3 100 5 0 12 0

 11 30 30 1
```

Grid Generation

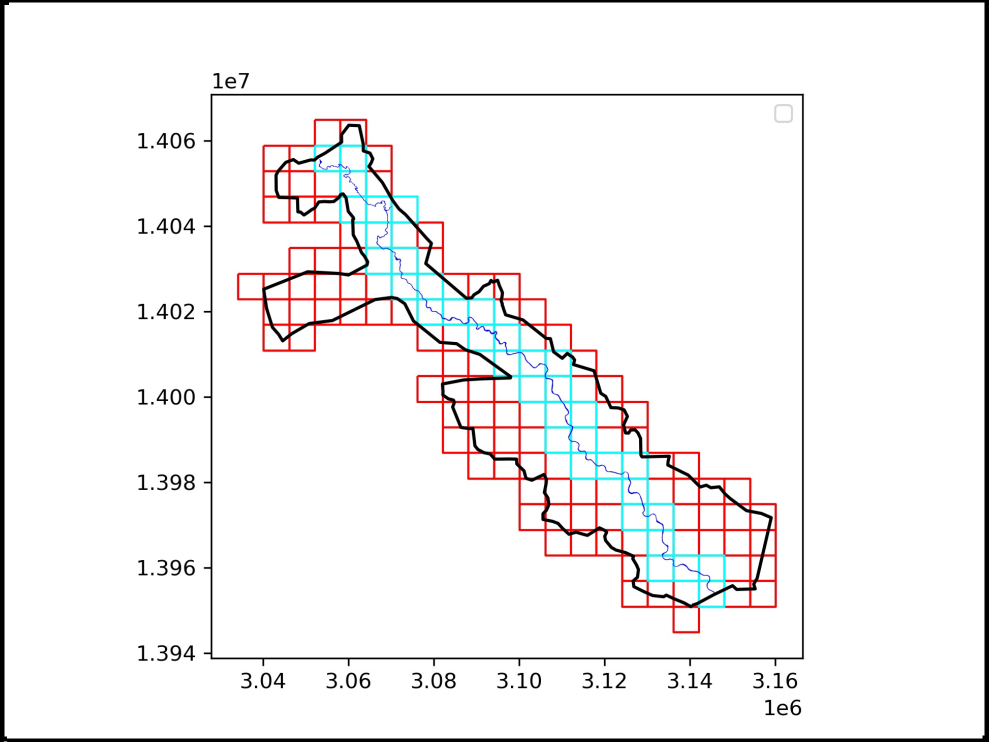
- The first step is to discretize the physical domain and find the associated grid connections

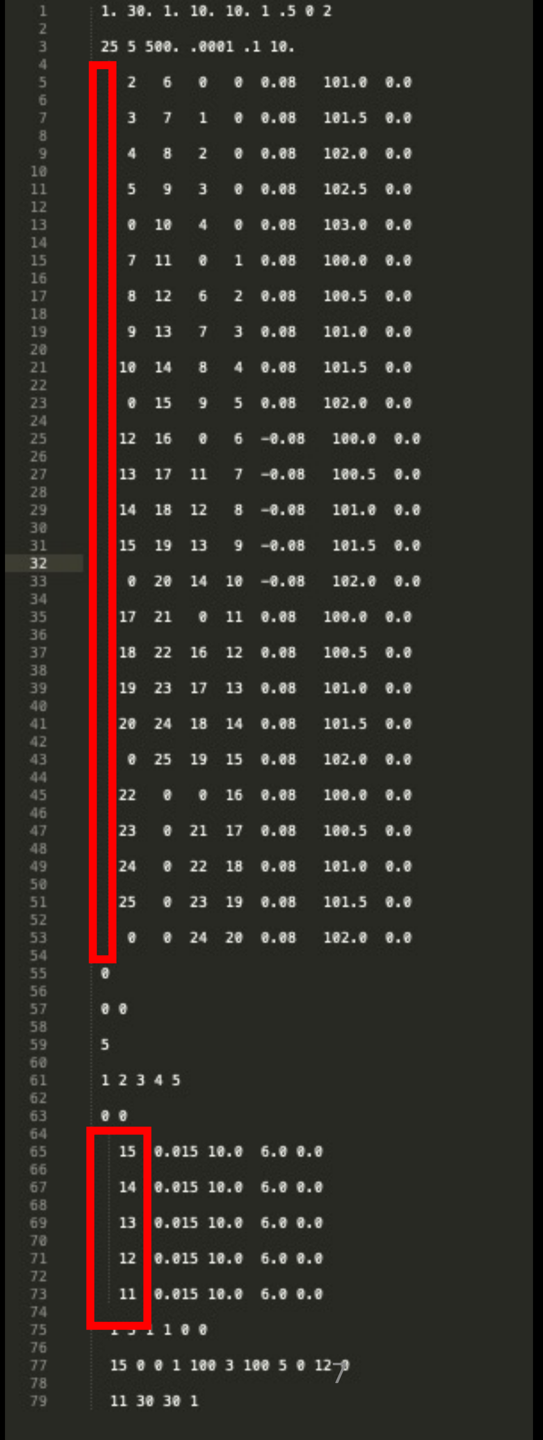
- Start with a shape file defining the floodplain region

- Using GeoPandas, create a georeferenced MxN grid fully encompassing the region

- Using GeoPandas intersection() function delete all cells not intersecting with the region

- Similarly, identify the channel cells, by flagging cells that intersect with the river

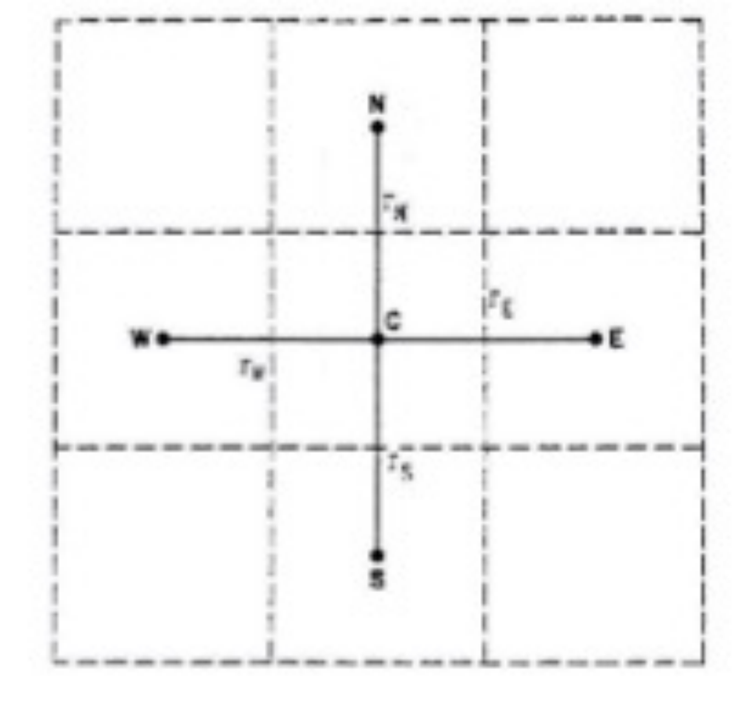




```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

  2   6   0   0  0.08   101.0  0.0

  3   7   1   0  0.08   101.5  0.0

  4   8   2   0  0.08   102.0  0.0

  5   9   3   0  0.08   102.5  0.0

  0  10   4   0  0.08   103.0  0.0

  7  11   0   1  0.08   100.0  0.0

  8  12   6   2  0.08   100.5  0.0

  9  13   7   3  0.08   101.0  0.0

 10  14   8   4  0.08   101.5  0.0

  0  15   9   5  0.08   102.0  0.0

 12  16   0   6  -0.08   100.0  0.0

 13  17  11   7  -0.08   100.5  0.0

 14  18  12   8  -0.08   101.0  0.0

 15  19  13   9  -0.08   101.5  0.0

  0  20  14  10  -0.08   102.0  0.0

 17  21   0  11  0.08   100.0  0.0

 18  22  16  12  0.08   100.5  0.0

 19  23  17  13  0.08   101.0  0.0

 20  24  18  14  0.08   101.5  0.0

  0  25  19  15  0.08   102.0  0.0

 22   0   0  16  0.08   100.0  0.0

 23   0  21  17  0.08   100.5  0.0

 24   0  22  18  0.08   101.0  0.0

 25   0  23  19  0.08   101.5  0.0

  0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

  15  0.015 10.0  6.0 0.0

  14  0.015 10.0  6.0 0.0

  13  0.015 10.0  6.0 0.0

  12  0.015 10.0  6.0 0.0

  11  0.015 10.0  6.0 0.0

 1 3 1 1 0 0

 15 0 0 1 100 3 100 5 0 12 0

 11 30 30 1
```

Grid Connections

- Once the grid is defined, we need to identify all the grid connections
- The grid connections are described by the grid ID numbers at the four cardinal directions (North, East, South, West)
- First use GeoPandas to compute and store the centroids of each cell
- Loop through all cells:
  - If the cells share a centroid coordinate (either X or Y), they **might** be neighbors (either North/South or East/West)
  - Neighbors are confirmed if the centroids are one cell width apart
  - The full grid information is updated and stored in a Pandas DataFrame




```
1. 30. 1. 10. 10. 1 .5 0 2
25 5 500. .0001 .1 10.
  2   6   0   0  0.08   101.0  0.0
  3   7   1   0  0.08   101.5  0.0
  4   8   2   0  0.08   102.0  0.0
  5   9   3   0  0.08   102.5  0.0
  0  10   4   0  0.08   103.0  0.0
  7  11   0   1  0.08   100.0  0.0
  8  12   6   2  0.08   100.5  0.0
  9  13   7   3  0.08   101.0  0.0
 10  14   8   4  0.08   101.5  0.0
  0  15   9   5  0.08   102.0  0.0
 12  16   0   6  -0.08   100.0  0.0
 13  17  11   7  -0.08   100.5  0.0
 14  18  12   8  -0.08   101.0  0.0
 15  19  13   9  -0.08   101.5  0.0
  0  20  14  10  -0.08   102.0  0.0
 17  21   0  11  0.08   100.0  0.0
 18  22  16  12  0.08   100.5  0.0
 19  23  17  13  0.08   101.0  0.0
 20  24  18  14  0.08   101.5  0.0
  0  25  19  15  0.08   102.0  0.0
 22   0   0  16  0.08   100.0  0.0
 23   0  21  17  0.08   100.5  0.0
 24   0  22  18  0.08   101.0  0.0
 25   0  23  19  0.08   101.5  0.0
  0   0  24  20  0.08   102.0  0.0
0
0 0
5
1 2 3 4 5
0 0
  15  0.015 10.0  6.0 0.0
  14  0.015 10.0  6.0 0.0
  13  0.015 10.0  6.0 0.0
  12  0.015 10.0  6.0 0.0
  11  0.015 10.0  6.0 0.0
 1 5 1 1 0 0
 15 0 0 1 100 3 100 5 0 12 0
 11 30 30 1
```

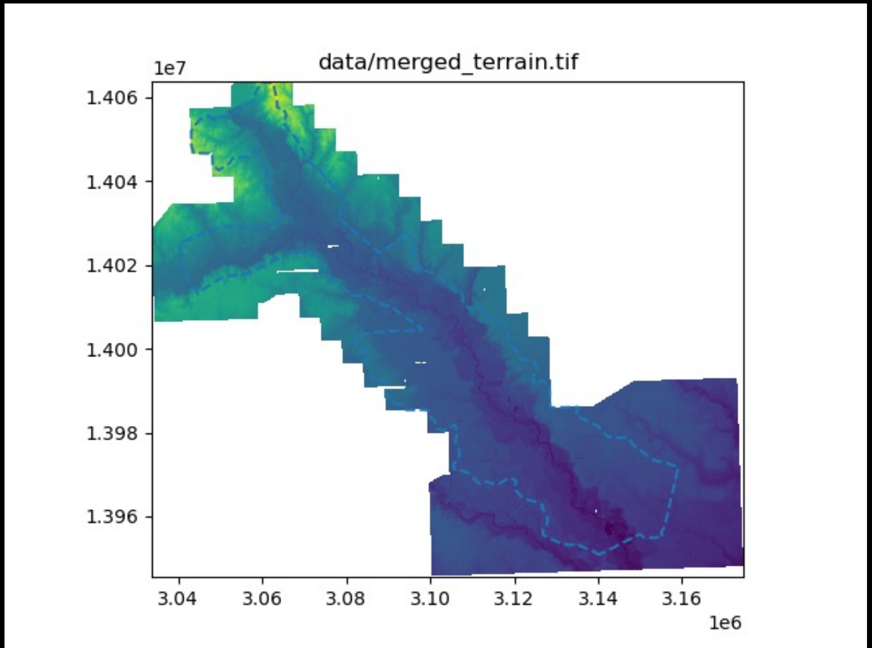
Terrain Elevation Data

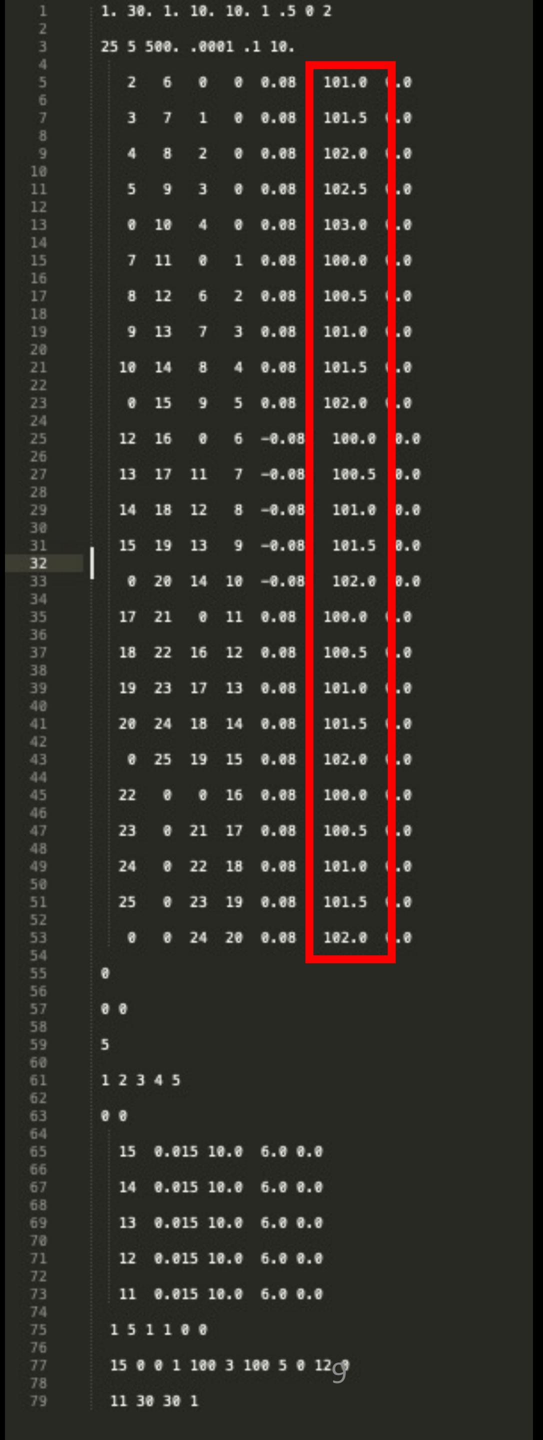
- After the grid is fully formed and connected, we need to assign an elevation for each grid cell

- The elevation data is typically contained in georeferenced raster files

- If the region is large, the data might be contained in multiple raster files (can become unwieldy)

- Use Rasterio to merge all of the raster files into one combined file



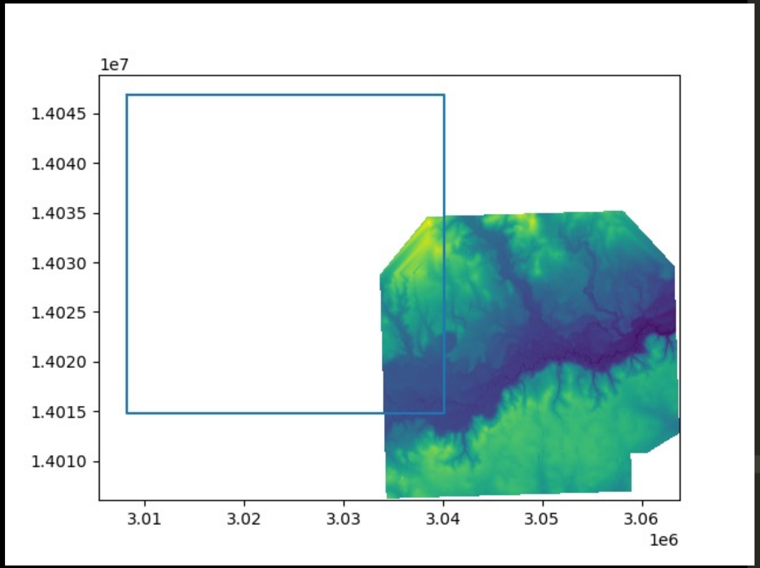


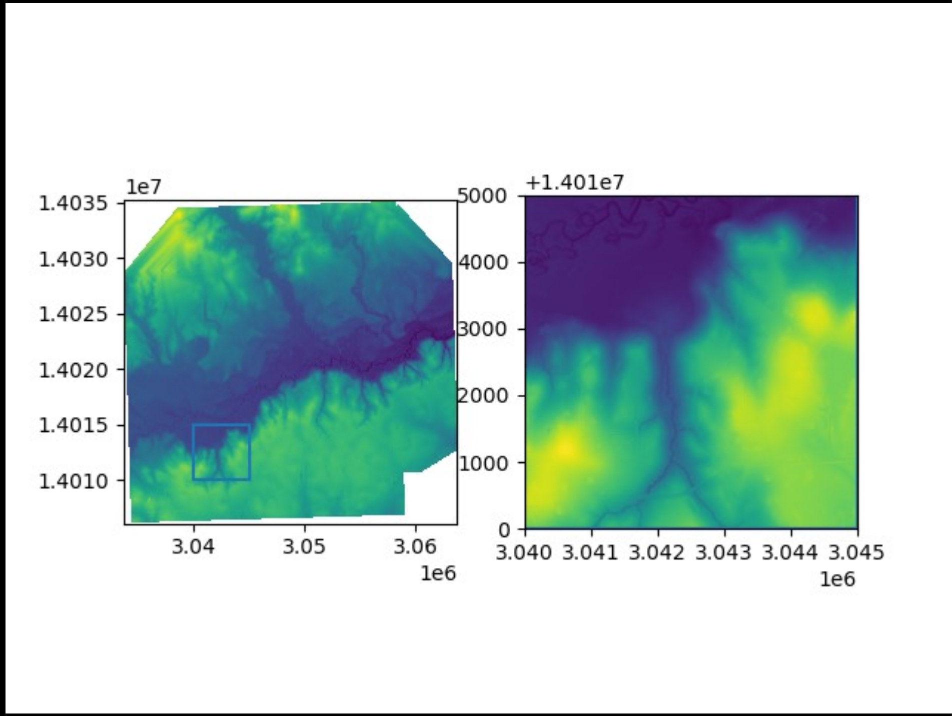
```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

   2   6   0   0  0.08   101.0  0.0

   3   7   1   0  0.08   101.5  0.0

   4   8   2   0  0.08   102.0  0.0

   5   9   3   0  0.08   102.5  0.0

   0  10   4   0  0.08   103.0  0.0

   7  11   0   1  0.08   100.0  0.0

   8  12   6   2  0.08   100.5  0.0

   9  13   7   3  0.08   101.0  0.0

  10  14   8   4  0.08   101.5  0.0

   0  15   9   5  0.08   102.0  0.0

  12  16   0   6 -0.08   100.0 0.0

  13  17  11   7 -0.08   100.5 0.0

  14  18  12   8 -0.08   101.0 0.0

  15  19  13   9 -0.08   101.5 0.0

   0  20  14  10 -0.08   102.0 0.0

  17  21   0  11  0.08   100.0  0.0

  18  22  16  12  0.08   100.5  0.0

  19  23  17  13  0.08   101.0  0.0

  20  24  18  14  0.08   101.5  0.0

   0  25  19  15  0.08   102.0  0.0

  22   0   0  16  0.08   100.0  0.0

  23   0  21  17  0.08   100.5  0.0

  24   0  22  18  0.08   101.0  0.0

  25   0  23  19  0.08   101.5  0.0

   0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

  15  0.015 10.0  6.0 0.0

  14  0.015 10.0  6.0 0.0

  13  0.015 10.0  6.0 0.0

  12  0.015 10.0  6.0 0.0

  11  0.015 10.0  6.0 0.0

 1 5 1 1 0 0

 15 0 0 1 100 3 100 5 0 120

 11 30 30 1
```

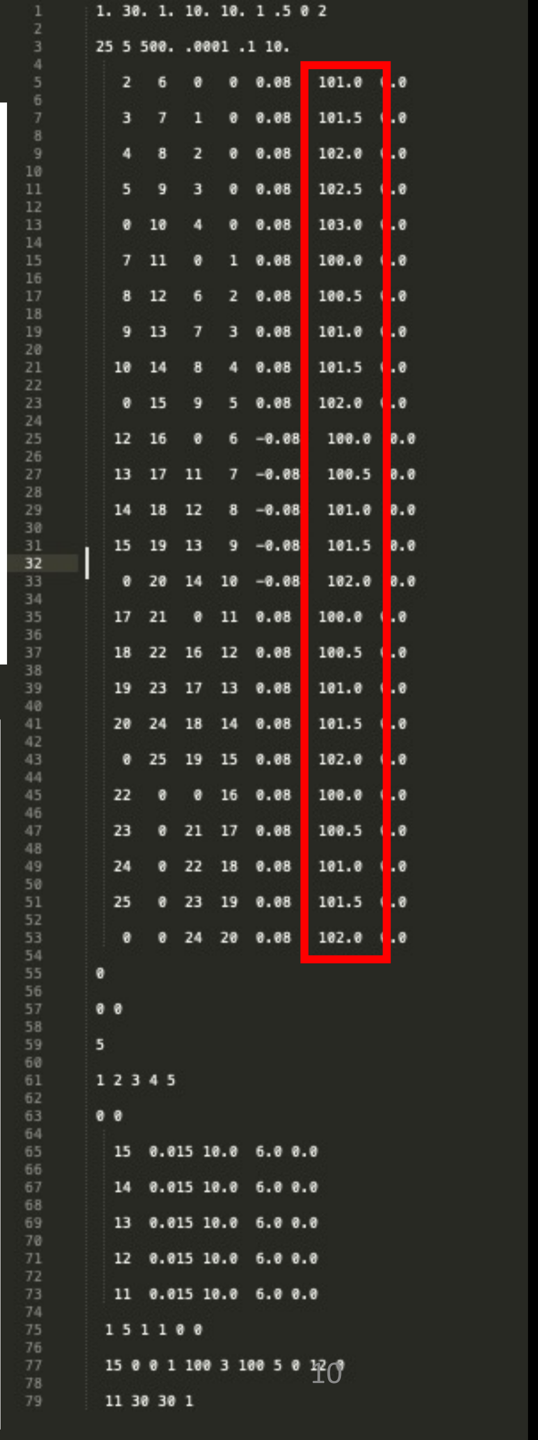
Terrain Elevation Data

- Now we have a single raster image containing all the elevation data

- Loop over every grid cell:
  - The grid cells will not necessarily be fully filled with raster data
  - Use Rasterio to "mask" the terrain data by the grid cell and average over the cell
  - Store this value in the Pandas DataFrame alongside the grid connection data







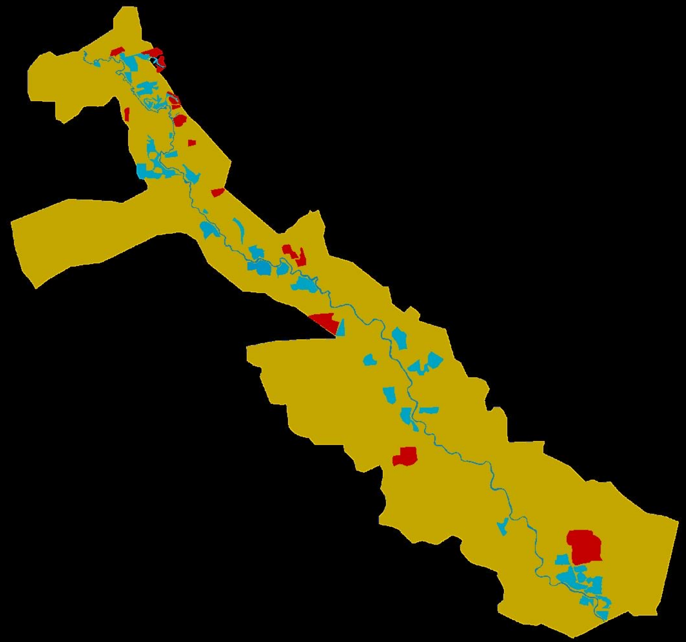
```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

  2   6   0   0  0.08   101.0  .0

  3   7   1   0  0.08   101.5  .0

  4   8   2   0  0.08   102.0  .0

  5   9   3   0  0.08   102.5  .0

  0  10   4   0  0.08   103.0  .0

  7  11   0   1  0.08   100.0  .0

  8  12   6   2  0.08   100.5  .0

  9  13   7   3  0.08   101.0  .0

 10  14   8   4  0.08   101.5  .0

  0  15   9   5  0.08   102.0  .0

 12  16   0   6  -0.08   100.0 0.0

 13  17  11   7  -0.08   100.5 0.0

 14  18  12   8  -0.08   101.0 0.0

 15  19  13   9  -0.08   101.5 0.0

  0  20  14  10  -0.08   102.0 0.0

 17  21   0  11  0.08   100.0  .0

 18  22  16  12  0.08   100.5  .0

 19  23  17  13  0.08   101.0  .0

 20  24  18  14  0.08   101.5  .0

  0  25  19  15  0.08   102.0  .0

 22   0   0  16  0.08   100.0  .0

 23   0  21  17  0.08   100.5  .0

 24   0  22  18  0.08   101.0  .0

 25   0  23  19  0.08   101.5  .0

  0   0  24  20  0.08   102.0  .0

0

0 0

5

1 2 3 4 5

0 0

 15  0.015 10.0  6.0 0.0

 14  0.015 10.0  6.0 0.0

 13  0.015 10.0  6.0 0.0

 12  0.015 10.0  6.0 0.0

 11  0.015 10.0  6.0 0.0

 1 5 1 1 0 0

 15 0 0 1 100 3 100 5 0 12 0

 11 30 30 1
```

Miscellaneous Grid Inputs

- Manning n values are defined by shape files defining each region with a corresponding manning region
- Use GeoPandas to fill the manning n values into the DataFrame by identifying the intersection of the grids and the manning polygons

- Initial water surface elevation is usually set to zero, but can be filled via raster file similar to the elevation

- The Pandas DataFrame now contains the grid connections, the manning value, the elevation, and the initial water surface elevation



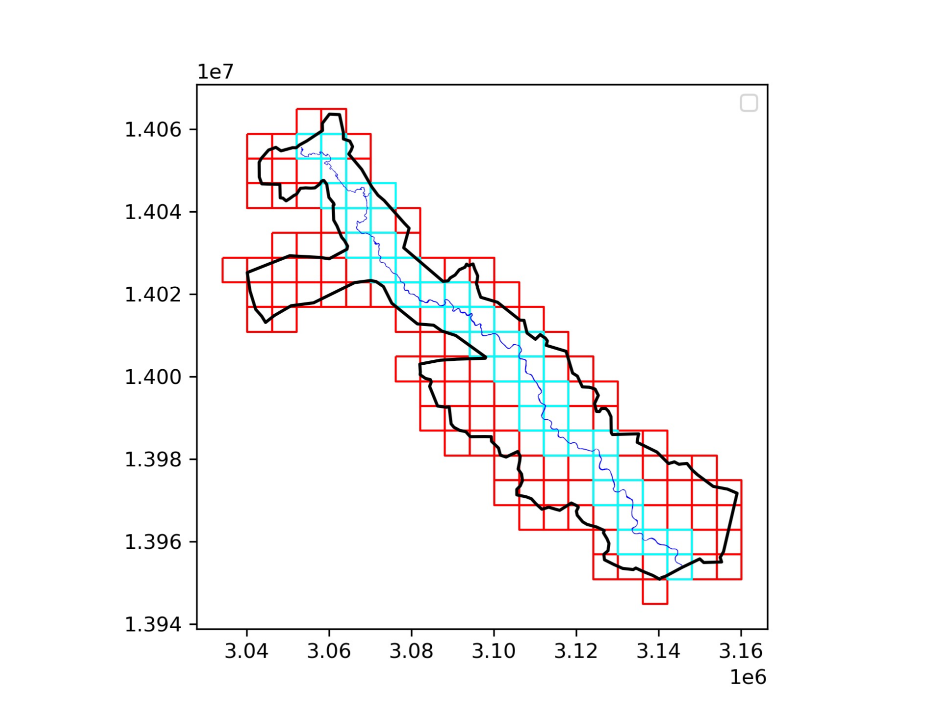
```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

  2   6   0   0  0.08   101.0  0.0

  3   7   1   0  0.08   101.5  0.0

  4   8   2   0  0.08   102.0  0.0

  5   9   3   0  0.08   102.5  0.0

  0  10   4   0  0.08   103.0  0.0

  7  11   0   1  0.08   100.0  0.0

  8  12   6   2  0.08   100.5  0.0

  9  13   7   3  0.08   101.0  0.0

 10  14   8   4  0.08   101.5  0.0

  0  15   9   5  0.08   102.0  0.0

 12  16   0   6  -0.08   100.0  0.0

 13  17  11   7  -0.08   100.5  0.0

 14  18  12   8  -0.08   101.0  0.0

 15  19  13   9  -0.08   101.5  0.0

  0  20  14  10  -0.08   102.0  0.0

 17  21   0  11  0.08   100.0  0.0

 18  22  16  12  0.08   100.5  0.0

 19  23  17  13  0.08   101.0  0.0

 20  24  18  14  0.08   101.5  0.0

  0  25  19  15  0.08   102.0  0.0

 22   0   0  16  0.08   100.0  0.0

 23   0  21  17  0.08   100.5  0.0

 24   0  22  18  0.08   101.0  0.0

 25   0  23  19  0.08   101.5  0.0

  0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

  15  0.015 10.0  6.0 0.0

  14  0.015 10.0  6.0 0.0

  13  0.015 10.0  6.0 0.0

  12  0.015 10.0  6.0 0.0

  11  0.015 10.0  6.0 0.0

 1 5 1 1 0 0

 15 0 0 1 100 3 100 5 0 12 0

 11 30 30 1
```

Channel Grid Information

- We have previously identified the grid cells that contain the channel
- These cell IDs are stored in a separate Pandas DataFrame along with associated manning values
- Need to fill in the river width and depth:

- Read the river bathymetry raster file if separate from the terrain data, otherwise reuse elevation
- To compute river width:
- Use GeoPandas, to easily access the length and area of the shape defining the river
- Compute effective width of the channel as area/length

- Store this information in the channel DataFrame

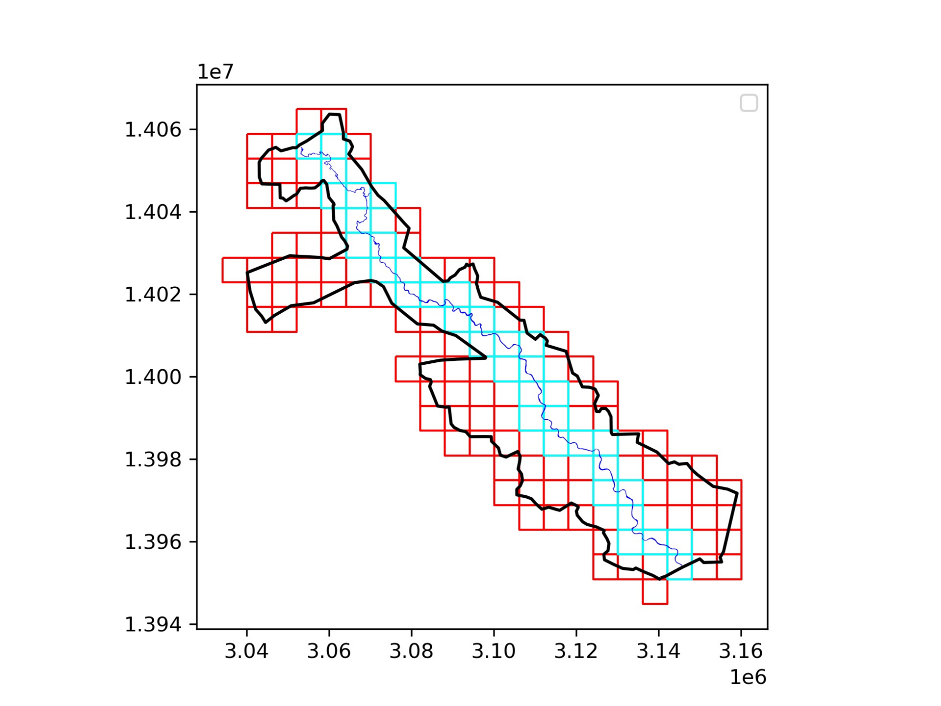


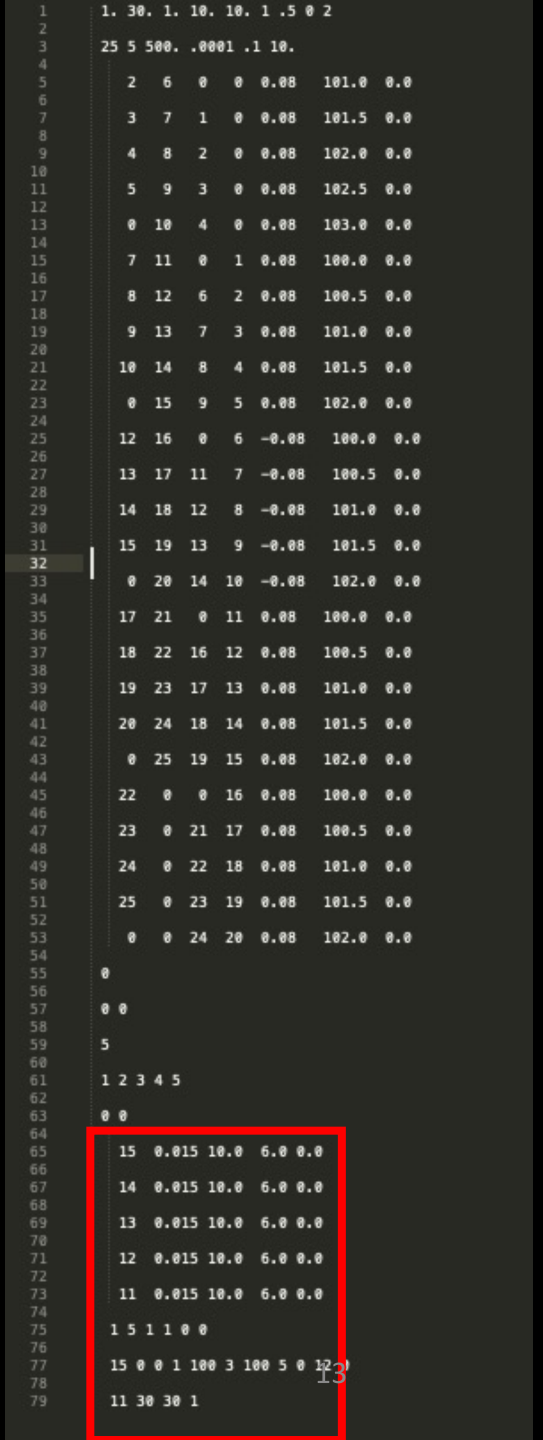

```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

   2   6   0   0  0.08   101.0  0.0

   3   7   1   0  0.08   101.5  0.0

   4   8   2   0  0.08   102.0  0.0

   5   9   3   0  0.08   102.5  0.0

   0  10   4   0  0.08   103.0  0.0

   7  11   0   1  0.08   100.0  0.0

   8  12   6   2  0.08   100.5  0.0

   9  13   7   3  0.08   101.0  0.0

  10  14   8   4  0.08   101.5  0.0

   0  15   9   5  0.08   102.0  0.0

  12  16   0   6  -0.08   100.0  0.0

  13  17  11   7  -0.08   100.5  0.0

  14  18  12   8  -0.08   101.0  0.0

  15  19  13   9  -0.08   101.5  0.0

   0  20  14  10  -0.08   102.0  0.0

  17  21   0  11  0.08   100.0  0.0

  18  22  16  12  0.08   100.5  0.0

  19  23  17  13  0.08   101.0  0.0

  20  24  18  14  0.08   101.5  0.0

   0  25  19  15  0.08   102.0  0.0

  22   0   0  16  0.08   100.0  0.0

  23   0  21  17  0.08   100.5  0.0

  24   0  22  18  0.08   101.0  0.0

  25   0  23  19  0.08   101.5  0.0

   0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

  15  0.015 10.0  6.0 0.0

  14  0.015 10.0  6.0 0.0

  13  0.015 10.0  6.0 0.0

  12  0.015 10.0  6.0 0.0

  11  0.015 10.0  6.0 0.0


 15 0 0 1 100 3 100 5 0 12 0

 11 30 30 1
```

Miscellaneous Channel Inputs

- The initial depth is set to zero, but can also be read in through a raster
- The channel inflow cell and outflow cell must be identified by finding the cell that contains the head and tail of the river
- The head and tail of the river are flagged as Inflow/Outflow boundary conditions in the DHM calculation

- At the inflow grid cell, we specify the inflow hydrograph (specifying the flow rate as a function of time)
- This can be read in as a text file or excel spreadsheet
- The outflow cell does not need a hydrograph, instead the outflow is approximated using a critical depth assumption





```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

  2   6   0   0  0.08   101.0  0.0

  3   7   1   0  0.08   101.5  0.0

  4   8   2   0  0.08   102.0  0.0

  5   9   3   0  0.08   102.5  0.0

  0  10   4   0  0.08   103.0  0.0

  7  11   0   1  0.08   100.0  0.0

  8  12   6   2  0.08   100.5  0.0

  9  13   7   3  0.08   101.0  0.0

 10  14   8   4  0.08   101.5  0.0

  0  15   9   5  0.08   102.0  0.0

 12  16   0   6  -0.08   100.0  0.0

 13  17  11   7  -0.08   100.5  0.0

 14  18  12   8  -0.08   101.0  0.0

 15  19  13   9  -0.08   101.5  0.0

  0  20  14  10  -0.08   102.0  0.0

 17  21   0  11  0.08   100.0  0.0

 18  22  16  12  0.08   100.5  0.0

 19  23  17  13  0.08   101.0  0.0

 20  24  18  14  0.08   101.5  0.0

  0  25  19  15  0.08   102.0  0.0

 22   0   0  16  0.08   100.0  0.0

 23   0  21  17  0.08   100.5  0.0

 24   0  22  18  0.08   101.0  0.0

 25   0  23  19  0.08   101.5  0.0

  0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

  15  0.015 10.0  6.0 0.0

  14  0.015 10.0  6.0 0.0

  13  0.015 10.0  6.0 0.0

  12  0.015 10.0  6.0 0.0

  11  0.015 10.0  6.0 0.0

 1 5 1 1 0 0

 15 0 0 1 100 3 100 5 0 12[illegible]

 11 30 30 1
```

Writing the DHM Input File

- We now have everything we need to write the DHM input file conveniently stored in Pandas DataFrames!!
- A subroutine takes these dataframes as input and writes a special fixed formatted file (this is important, because the Fortran code expects a specific format)
- Use Python f-strings to make writing formatted strings easy

```python
# BLOCK 1
outfile.write(
    f" {dtmin} {dtmax} {dti} {dtd} {simul} {num_iter} {tout} {kode} {kmodel}\n\n"
)
# BLOCK 2
outfile.write(f" {nnod} {nodc} {side} {tol} {dtol} {dtolp} \n \n")
# BLOCK 3
for idx, cell in fp_grid.iterrows():
    outfile.write(
        f"""{str(cell["North"]).rjust(5)} {str(cell["East"]).rjust(4)} {str(cell["So
    )
# BLOCK 4
outfile.write(f" {neri}\n\n")
# BLOCK 5
outfile.write(f" {nfpi} {npfpi}\n\n")
# BLOCK 6
outfile.write(f""" {len(fp_grid[fp_grid["West"] == 0])}\n\n""")
# BLOCK 7
for idx, _ in (fp_grid[fp_grid["West"] == 0]).iterrows():
    outfile.write(f""" {idx} """)
outfile.write("\n\n")
```

```
1. 30. 1. 10. 10. 1 .5 0 2

25 5 500. .0001 .1 10.

   2   6   0   0  0.08   101.0  0.0

   3   7   1   0  0.08   101.5  0.0

   4   8   2   0  0.08   102.0  0.0

   5   9   3   0  0.08   102.5  0.0

   0  10   4   0  0.08   103.0  0.0

   7  11   0   1  0.08   100.0  0.0

   8  12   6   2  0.08   100.5  0.0

   9  13   7   3  0.08   101.0  0.0

  10  14   8   4  0.08   101.5  0.0

   0  15   9   5  0.08   102.0  0.0

  12  16   0   6  -0.08   100.0  0.0

  13  17  11   7  -0.08   100.5  0.0

  14  18  12   8  -0.08   101.0  0.0

  15  19  13   9  -0.08   101.5  0.0

   0  20  14  10  -0.08   102.0  0.0

  17  21   0  11  0.08   100.0  0.0

  18  22  16  12  0.08   100.5  0.0

  19  23  17  13  0.08   101.0  0.0

  20  24  18  14  0.08   101.5  0.0

   0  25  19  15  0.08   102.0  0.0

  22   0   0  16  0.08   100.0  0.0

  23   0  21  17  0.08   100.5  0.0

  24   0  22  18  0.08   101.0  0.0

  25   0  23  19  0.08   101.5  0.0

   0   0  24  20  0.08   102.0  0.0

0

0 0

5

1 2 3 4 5

0 0

  15  0.015 10.0  6.0 0.0

  14  0.015 10.0  6.0 0.0

  13  0.015 10.0  6.0 0.0

  12  0.015 10.0  6.0 0.0

  11  0.015 10.0  6.0 0.0

 1 5 1 1 0 0

 15 0 0 1 100 3 100 5 0 12 0

 11 30 30 1
```

Running DHM

- The execution of DHM is very fast due to the lightweight Fortran code
- Output is saved in a readable, formatted manner
- While this improves humans readability, it makes computer readability more challenging
- Currently working on a post-processing script that will comb through the output and read results for automatic plotting and data analysis

```
    ***CHANNEL RESULTS***

       INFLOW  RATE AT NODE  20 IS EQUAL TO     5662.96
       OUTFLOW RATE AT NODE 125 IS EQUAL TO      722.34

  NODE          11          12          13          14          15          16
  17           18          19          20
  DEPTH        0.000       0.000       0.000       0.000       0.000       0.000
  0.000        0.000       0.000      23.056
ELEVATION      0.000       0.000       0.000       0.000       0.000       0.000
0.000        0.000       0.000     179.956

  NODE          21          22          23          24          25          26
  27           28          29          30
  DEPTH        0.000       0.000       0.000       0.000       0.000       0.148
  12.801        0.000       0.000       0.000
ELEVATION      0.000       0.000       0.000       0.000       0.000     155.348
155.401      208.400       0.000       0.000

  NODE          31          32          33          34          35          36
  37           38          39          40
  DEPTH        0.000       0.097       2.356       0.024       0.000       0.000
  0.000        0.076       0.224       0.000
ELEVATION      0.000     130.997     131.056     138.824       0.000       0.000
0.000      120.676     124.024       0.000
```

```
 ENTER INPUT FILE NAME
 (Example:   DHM21.DAT)   ->

WFSJR.dat
 PRINTOUT OPTIONS:
       1= RESULTS SENT DIRECTLY TO PRINTER
       2= RESULTS SENT TO A FILE ON DISK
 SELECT DESIRED OPTION ->

2

 ENTER RESULTS FILE NAME
 (Example:   DHM21.RES)    ->

output.res
  MODEL TIME =     0.50 HOURS
  MODEL TIME =     1.00 HOURS
  MODEL TIME =     1.50 HOURS
  MODEL TIME =     2.00 HOURS
  MODEL TIME =     2.50 HOURS
  MODEL TIME =     3.00 HOURS
  MODEL TIME =     3.50 HOURS
  MODEL TIME =     4.00 HOURS
  MODEL TIME =     4.50 HOURS
  MODEL TIME =     5.00 HOURS
  MODEL TIME =     5.50 HOURS
  MODEL TIME =     6.00 HOURS
  MODEL TIME =     6.50 HOURS
```

Conclusions and Next Steps

- The Diffusion Hydrodynamic Model (DHM) is a fast, efficient code that has been repeatedly validated over the years
- Modern engineering and scientific codes are expected to include visualization tools or data structures that facilitate problem set up, visualization, and analysis
- We created a series of Python routines that will automatically form a DHM input file from standard GIS shape and raster files

DHM
Diffusion Hydrodynamic Model
USGS Water Resources Division Special Report 87-4137

http://www.diffusionhydrodynamicmodel.com/

https://github.com/nickwimer/pyDHM.git