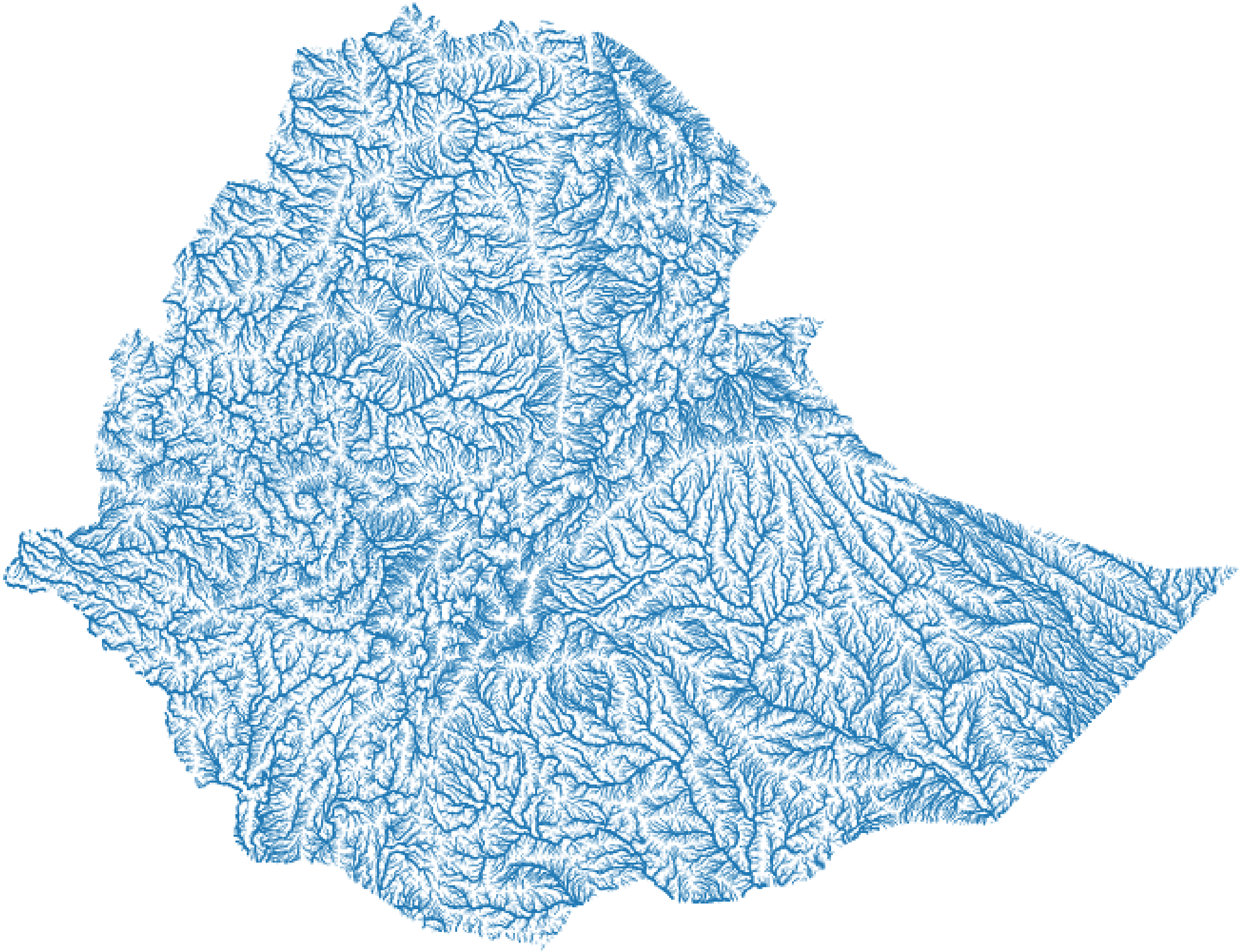
WRF-Hydro GIS Pre-processing Tools

2022-12-22

1 GIS Pre-processing Tools



The purpose of the WRF Hydro GIS Pre-Processing Tool is to building the geospatial input files for running a WRF-Hydro simulation.

- data layers for terrestrial overland flow,
- subsurface flow and
- channel routing processes required by WRF Hydro model.

The outputs from these tools are `geospatial` and `tabular` data layers.

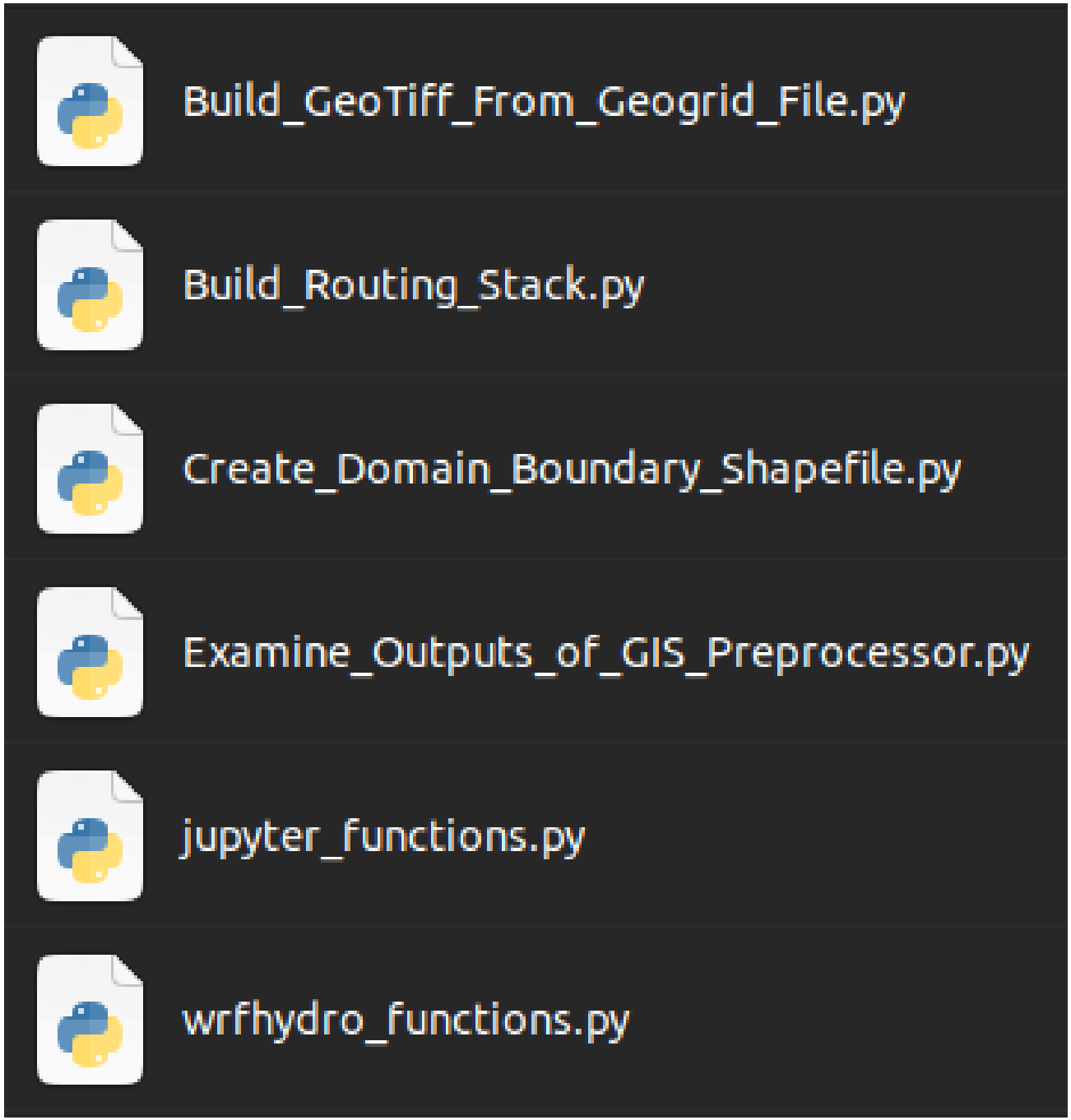
This workflow for creating WRF-Hydro routing grids can be accomplished using either `NCAR Python Scripts` or `ArcGIS Pre-processor`.

1.1 Method 1: NCAR Python Scripts

The **NCAR** has developed an open source **WRF-Hydro GIS Pre-Processor** modules to assist the preparation of hydrological and topographic inputs for WRF-Hydro models.

The NCAR WRF-Hydro GIS Pre-Processor module rely on several python packages and it consists about 14 scripts with different functionalities. All the scripts must be in one directory and rely on `wrfhydro functions.py`

These are the primary scripts we will employ in this training session.


Build_GeoTiff_From_Geogrid_File.py
Build_Routing_Stack.py
Create_Domain_Boundary_Shapefile.py
Examine_Outputs_of_GIS_Preprocessor.py
jupyter_functions.py
wrfhydro_functions.py

To create a stable working environment for the scripts `Miniconda` or `Anaconda` Python package mangers are recommended via conda tools.

To use these scripts properly, a separate
conda environment must be manually created using the following python packages.

The packages are saved in a file called `environment.yml` with the desired environment name, in this case `wrfh_gis_env2`.

Contents of the `environment.yml`:

```
name: wrfh_gis_env2
dependencies:
  - numpy
  - pandas
  - gdal
  - geopandas
  - ipympl
  - ipywidgets
  - jupyter
  - jupyterlab
  - matplotlib
  - netCDF4
  - nodejs
  - numpy
  - pandas
  - pyproj
  - rasterio
  - widgetsnbextension
  - pip
  - pip:
```

```
    - ipyleaflet
    - whitebox
```

To install the packages with all their dependencies within the `wrfh_gis_env2` environment, open your terminal and run the following command:

```
conda env create -f environment.yml
```

To activate the `wrfh_gis_env2` environment type:

```
conda activate wrfh_gis_env2
```

1.1.1 Defining paths for inputs and output files

```
# Import standard utility modules
import os     # For creating and removing a directory, fetching its
        contents, changing and identifying the current directory, e
        tc.
import shutil # Perform high-level operation like copy and create on
        files and collections of files.
```

Set GIS input data directory

```
# Define the directory
gis_data_folder = "$HOME/Documents/GIS_Training"

# Create the input data directory
!mkdir -p $HOME/Documents/GIS_Training

# If you are on the python/juppter notebook dont forget to use the e
        xclamation `!`
```

Change the directory to the input data and directory and get current working directory

```
os.chdir(gis_data_folder)
cwd = os.getcwd()
cwd
```

Set path to input data

```
data_folder = os.path.join(cwd, 'GIS_DATA')
data_folder
```

Set path to output data

```
output_folder = os.path.join(cwd, 'Outputs')
output_folder
```

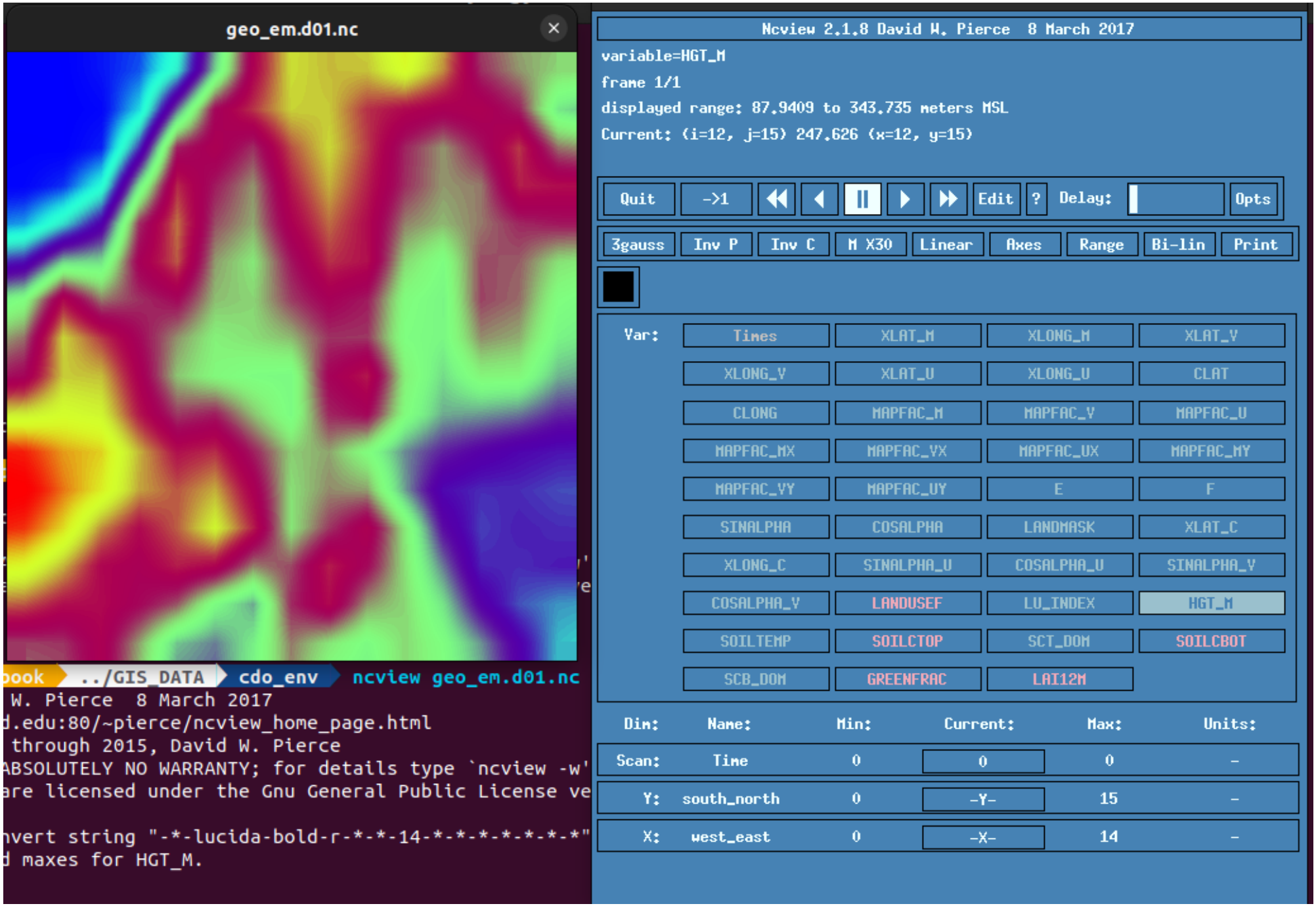
Clear all outputs from previous runs and re-creating the output directory

```
if os.path.exists(output_folder):
   shutil.rmtree(output_folder)
os.mkdir(output_folder)
```

1.1.2 Create domain boundary files

The `Create_Domain_Boundary_Shapefile.py` script is used to create a polygon shapefile that defines the boundary of the domain in projected coordinates. This rectangular domain is used as the routing domain by WRF-Hydro.

The main input file used in this script is the `geo em.d01.nc` produced from WPS output.



Let's define the path for the `geo_em.d01.nc`

```
in_geogrid = os.path.join(data_folder, 'geo_em.d01.nc')
in_geogrid
```

Run the script with required parameters

```
# Print information to screen for reference
print('Command to run:\n')
print('Python Create_Domain_Boundary_Shapefile.py \\\n\t -i {0} \\\n
        \t -o {1}\n'.format(in_geogrid, output_folder))

! python Create_Domain_Boundary_Shapefile.py -i {in_geogrid} -o {out
        put_folder}
```

Out put:

output_folder: `geo_em.d01_boundary.shp`

1.1.2.1 Visualize the domain boundary shapefile

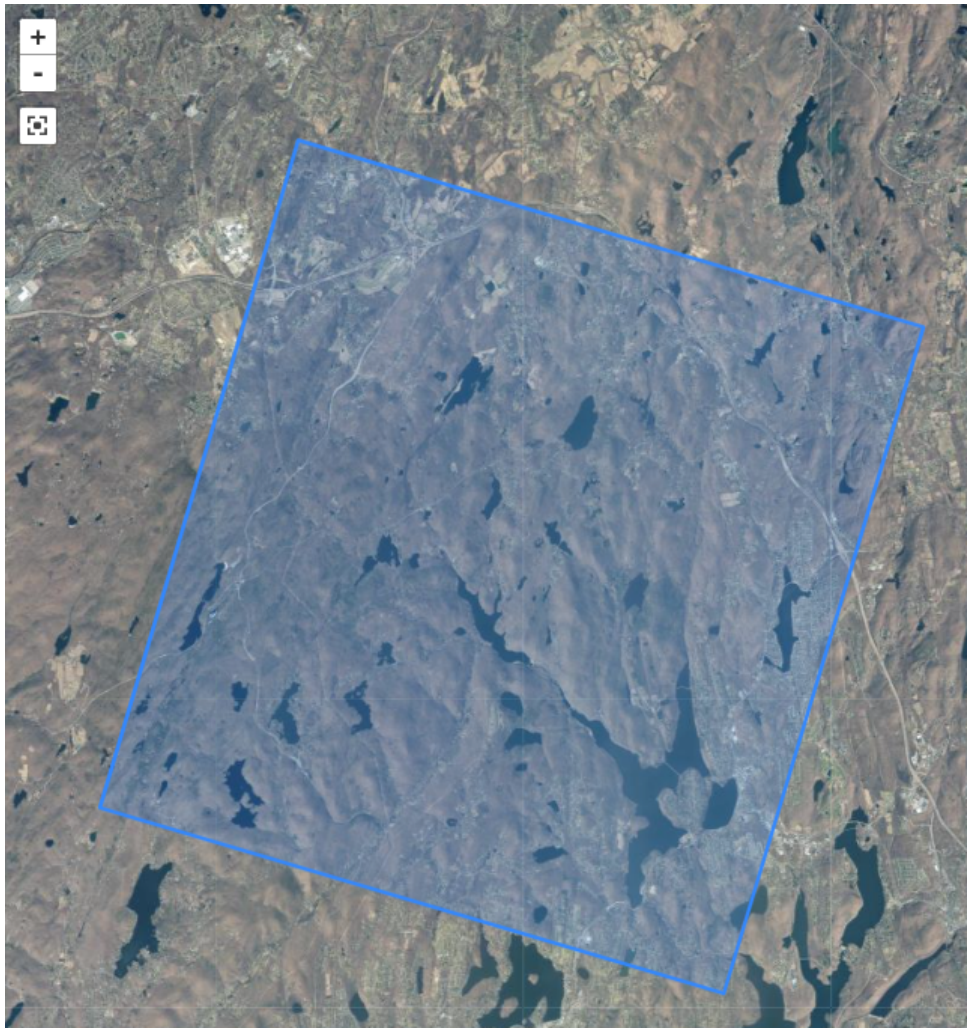
Now that the domain boundary shapefile has been created, we want to explore the domain.

```
import json
import geopandas
import ipyleaflet
from ipyleaflet import Map, GeoJSON, ScaleControl, FullScreenContro
        l, basemaps, SplitMapControl, basemap_to_tiles, LayersContr
        ol
from jupyter_functions import create_map

import warnings
warnings.filterwarnings("ignore")

# Setup display items
boundary_shp = os.path.join(output_folder,'geo_em.d01_boundary.shp')
b_shp = geopandas.read_file(boundary_shp)
b_shp = b_shp.to_crs(epsg=4326)

# Export vector to GeoJSON
b_json = os.path.join(output_folder, 'boundary.json')
b_shp.to_file(b_json, driver='GeoJSON')

# Read GeoJSON
with open(b_json, 'r') as f:
```

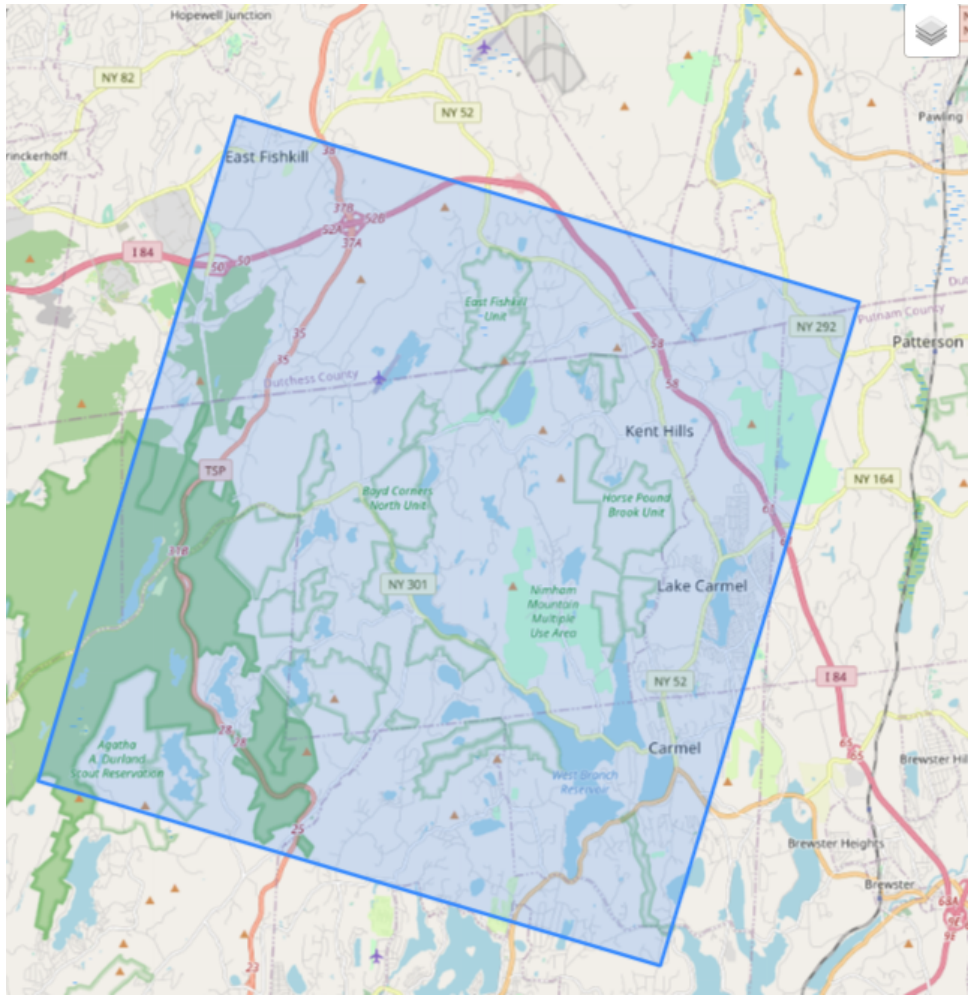
```
    data = json.load(f)

# Obtain vector center point
x = b_shp.geometry.centroid.x
y = b_shp.geometry.centroid.y
map_center = y[0], x[0]

# Instantiate map object
m = Map(center=(41.50, -73.73), zoom=10, scroll_wheel_zoom=True)

# Read GeoJSON
with open(b_json, 'r') as f:
    data = json.load(f)

# Obtain vector center point
x = b_shp.geometry.centroid.x
y = b_shp.geometry.centroid.y
map_center = y[0], x[0]

# Instantiate map object
m = create_map(map_center, 10)
```

```
# Read GeoJSON
geo_json = GeoJSON(data=data, name='Domain boundary')

# Define basemaps to swipe between
right_layer = basemap_to_tiles(basemap=basemaps.OpenStreetMap.Mapni
        k)
left_layer = basemap_to_tiles(basemap=basemaps.Esri.WorldImagery)

# Setup basemap swipe control
control = SplitMapControl(left_layer=left_layer, right_layer=right_l
        ayer)
m.add_control(control)
m.add_layer(geo_json)

# Draw map
m
```

Google Satellite Imagery View



Open Street Map View



1.1.3 Build GeoTiff raster from a surface elevation

The `Build_GeoTiff_From_Geogrid_File.py` script converts variables within the `Geogrid` or `Fulldom_hires` netCDF files into the `GeoTiff` raster file. Let's use the `HGT_M` variable (surface elevation).

```
# Define the variable to export to raster
in_var = "HGT_M"

# Define the output raster file using variable name defined above
out_file = os.path.join(output_folder, f'{in_var}.tif')

# Print information to screen for reference
print('Command to run:\n')
print('python Build_GeoTiff_From_Geogrid_File.py \\\n\t -i {0} \\\n
        \t -v {1} \\\n\t -o {2}\n'.format(in_geogrid, in_var, out_f
        ile))

# Run the script with required parameters
! python Build_GeoTiff_From_Geogrid_File.py -i {in_geogrid} -v {in_v
        ar} -o {out_file}
```

Output:

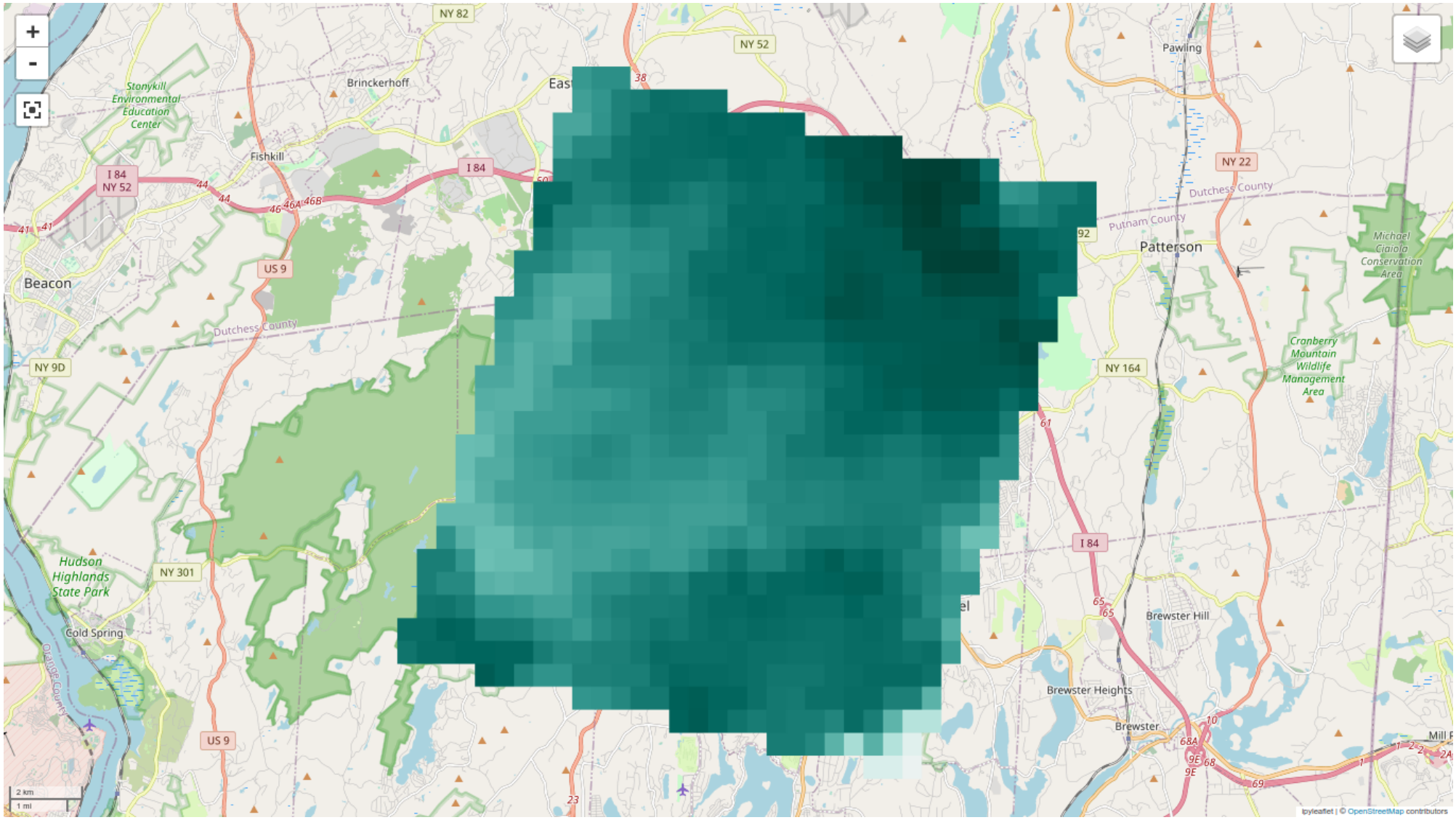
- Raster_Outputs_Prj
    - HGT_M.png
    - HGT_M.tif
- boundary.json
- HGT_M.tif

Draw the basemap

```
# Import Python visualization libraries

import rasterio
from matplotlib import pyplot
from osgeo import gdal
from ipyleaflet import ImageOverlay
from jupyter_functions import cmap_options, show_raster_map

# Create a map object from pre-build function
m2 = create_map(map_center, 10)

# Render the map
m2
```

Overlay the surface elevation on the basemap

```
show_raster_map(out_file, m2, b_shp, output_folder)
```



1.1.4 Building the hydrologic routing grids

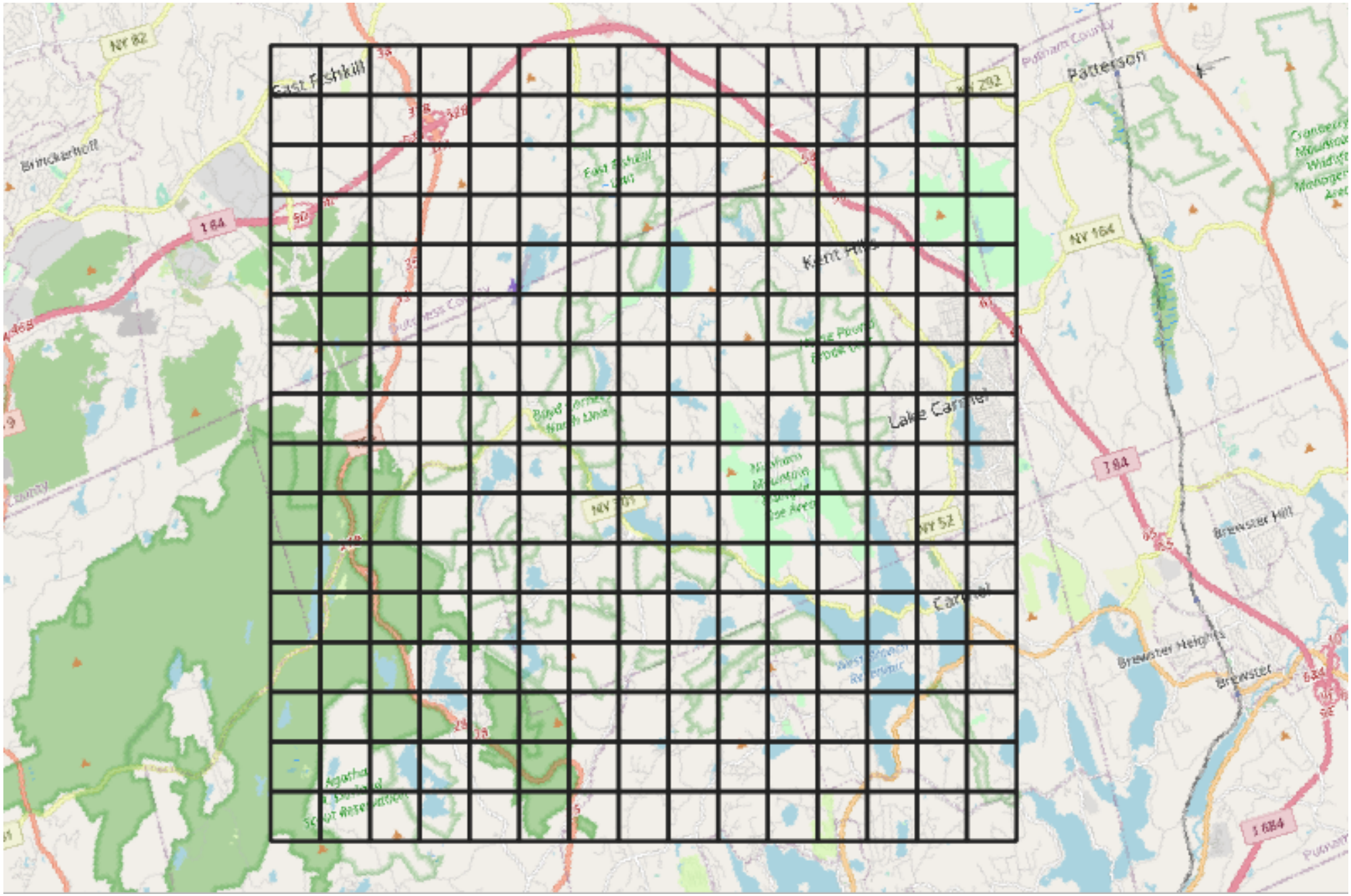
The `Build_Routing_Stack.py` script is used to build the full set of hydrological routing grids and additional data required by WRF-Hydro model. This is the main utility for performing WRF-Hydro GIS pre-processing.

The inputs are:

- `-i` : WPS GEOGRID file (geo_em.d0*.nc)

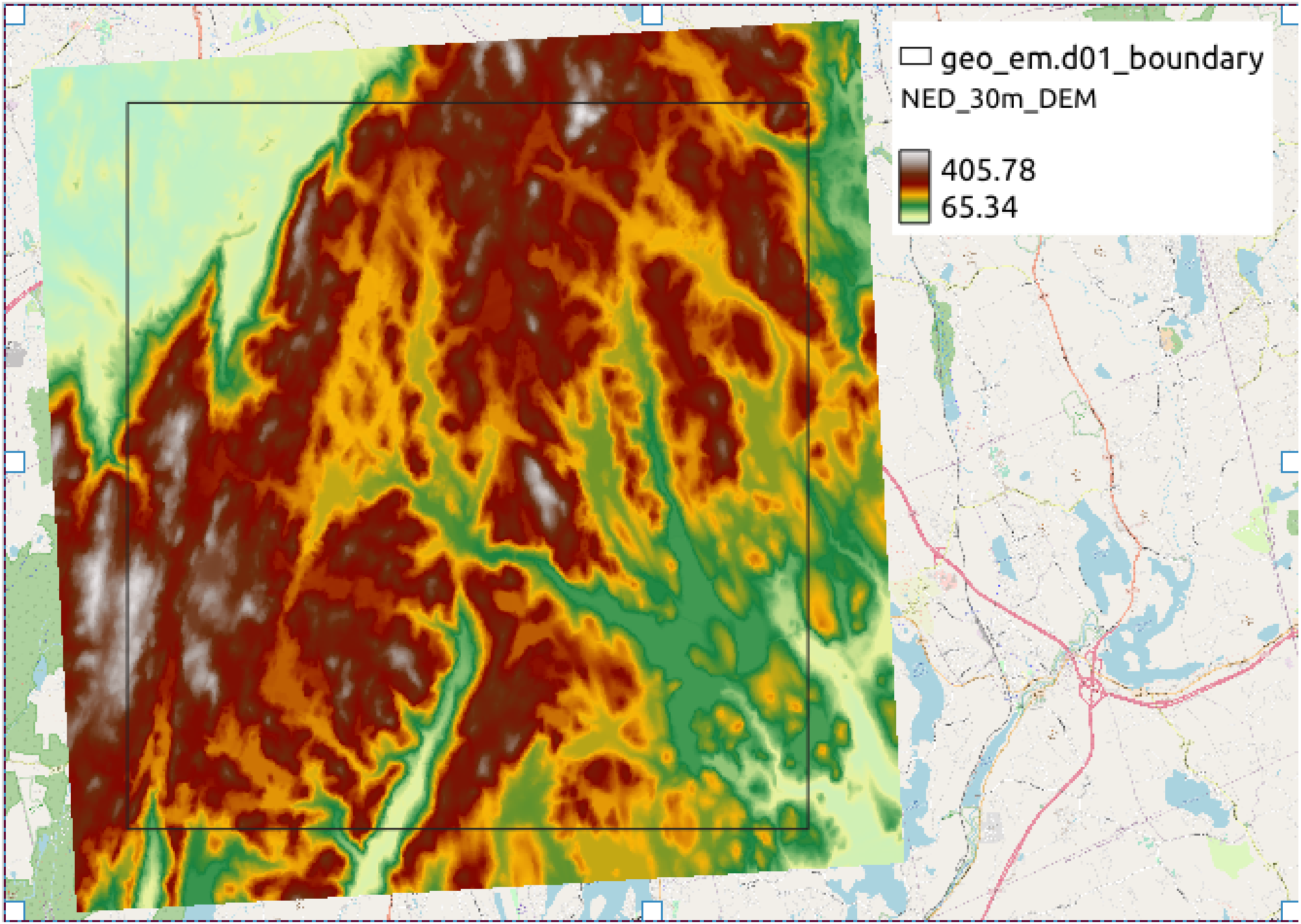
The purpose of the GEOGRID file in the WRF-Hydro GIS Pre-processor is to define

- the simulation domain
- coordinate reference system
- extent
- resolution
- interpolate various static geographical datasets to the model grid

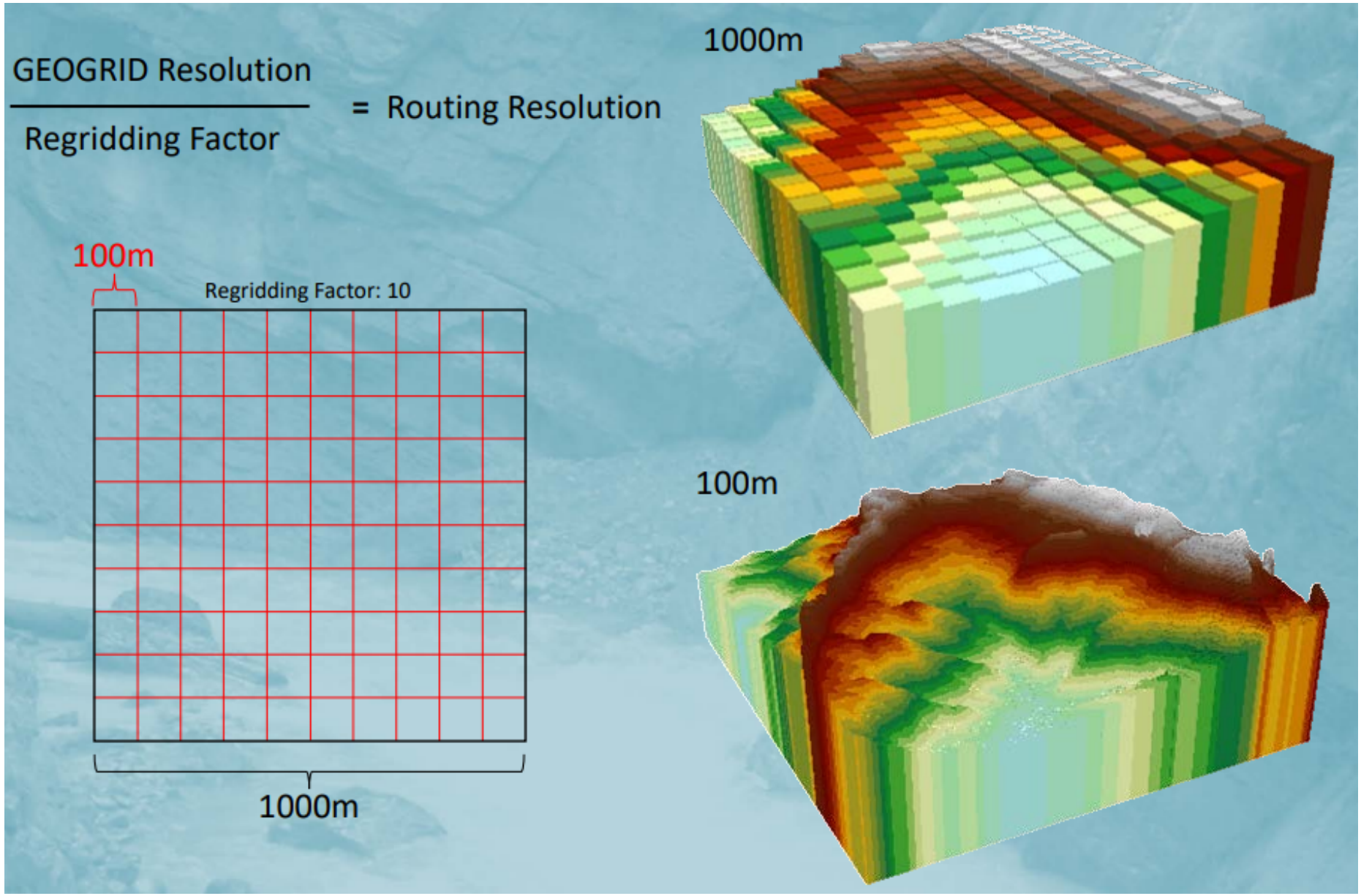


-d : High-resolution Elevation file (Esri GRID, GeoTIFF, VRT, etc.)

- Must be an ArcGIS-readable raster format
- Must contain valid coordinate reference system
- Must cover entire extent (and more) of your GEOGRID domain
- Elevation units must be converted to meters (m)
- Should by hydrologically corrected


geo_em.d01_boundary
NED_30m_DEM
405.78
65.34

- `-R` : Re-gridding factor



- `-t` : minimum number of routing grid cells to define the basin area and streams

- `--CSV` : Station locations file (.csv)

FID	LON	LAT	STATION	Site_No
1	-73.75913	41.46844	A	1374559
2	-73.73533	41.44862	B	1374581
3	-73.69407	41.41246	C	137462010

- `-b` : Option to mask channel grids not contributing to provided station locations
- `-r` : Reach based (Muskingum / Muskingum-Cunge) routing option
- `-l` : -Lake Polygons (.shp)
- `-O` : OVROUGHRTFAC: Multiplier on Manning's roughness for overland flow (default=1.0)
- `-T` : RETDEPRTFAC: Multiplier on maximum retention depth before flow is routed as overland flow (default=1.0)
- `LKSATFAC` : Multiplier on saturated hydraulic conductivity in lateral flow direction. default=1000.0

- `--starts` : Path to point shapefile containing channel initiation locations (overrides `-t` parameter).
- `--gw` : Path to polygon shapefile containing groundwater basin locations

The output will be stored in zip file with WRF-Hydro domain and parameter files.

- `-o` : Output ZIP File containing all script outputs

Let's import the script and define the input file paths

```
import Build_Routing_Stack

# Define script input parameters using python variables
in_geogrid = os.path.join(data_folder, 'geo_em.d01.nc')
lakes = os.path.join(data_folder, 'lake_shapes', 'lakes.shp')
csv = os.path.join(data_folder, 'forecast_points.csv')
in_dem = os.path.join(data_folder, 'NED_30m_DEM.tif')
regrid_factor = 4
routing_cells = 25
out_zip = os.path.join(output_folder, 'Gridded_test.zip')
```

Execute the routing scrip

```
# Print information to screen for reference
print('Command to run:\n')
print('python Build_Routing_Stack.py \\\n\t -i {0} \\\n\t -l {1}
        \\\n\t --CSV {2} \\\n\t -d {3} \\\n\t -R {4} \\\n\t -t {5}
        \\\n\t -o {6}\n'.format(in_geogrid, lakes, csv, in_dem, reg
        rid_factor, routing_cells, out_zip))


# Run the script with required parameters
! python Build_Routing_Stack.py -i {in_geogrid} \
-l {lakes} \
--CSV {csv} \
-d {in_dem} \
-R {regrid_factor} \
-t {routing_cells} \
-o {out_zip}
```

Understanding the outputs:

The `Build_Routing_Stack.py` script creates a Zip archive of output files according to the options provided to the tool. There will be at least four netCDF files.

- `Fulldom_hires.nc` : It includes the main input data for the routing.

Name	Description
CHANNELGRID	Channel pixels = 0, non-channel pixels = -9999
FLOWDIRECTION	This grid gives the direction of flow using the D8 algorithm
FLOWACC	Gives the number of contributing cells for each cell in the domain
TOPOGRAPHY	Elevation grid
RETDEPRTFAC	Retention depth before flow
OVROUGHRTFAC	Manning’s roughness for overland flow
STREAMORDER	Stream order grid, calculated using the Strahler method (Strahler 1957).
frxst_pts	Gage location
basn_msk	Basins grid

Name	Description
LAKEGRID	The lake grid. If a lake polygon shapefile is provided to the `-l` parameter, this grid will contain ID values for each lake that can be resolved on the routing grid.
landuse	Landuse resampled using Nearest Neighbor to the resolution of the routing grid.
LKSATFAC	Saturated hydraulic conductivity in lateral flow direction

- `GEOGRID_LDASOUT_Spatial_Metadata.nc` : It provides the spatial metadata to the land surface model output.
- `GWBASINS.nc` : Location of ground water basins regridded to the LSM grid resolution.
- `GWBUCKPARM.nc` : 1D groundwater basin parameter files (Basin ID, Basin area, Basin height, etc)

- `LAKEPARML` : 1D Lake parameters files: (Lake ID, Lake area, Maximum lake elevation, lake centroid etc.)
- `lakes.shp`

1.1.5 Visualize the hydrologic routing grids

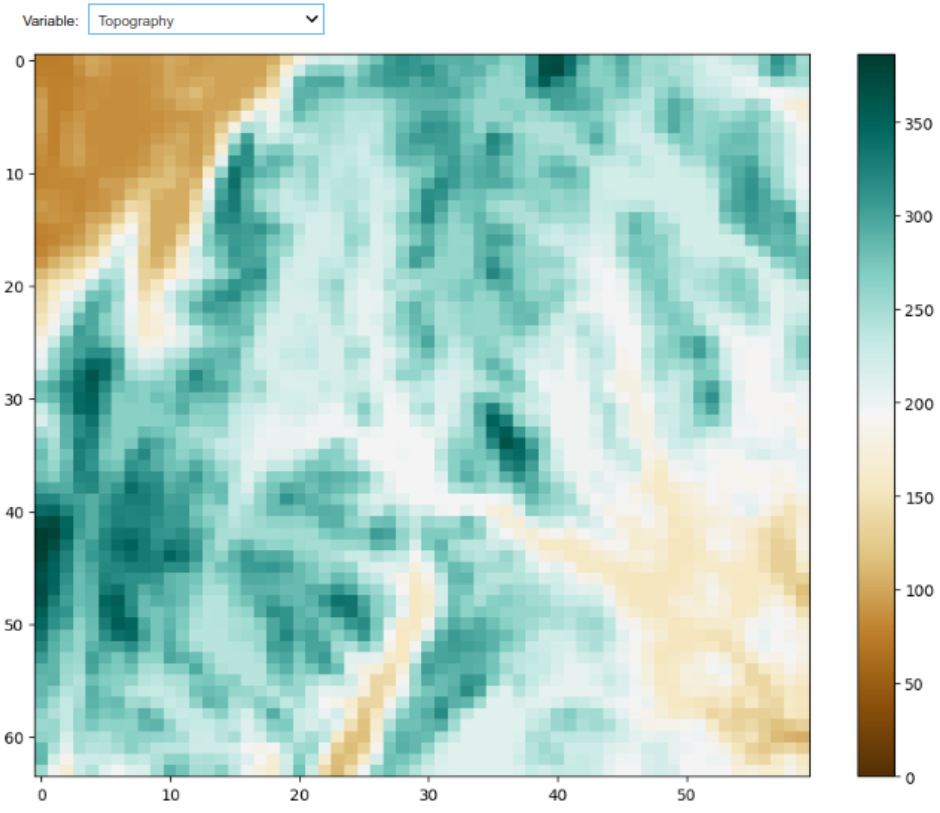
The `Examine_Outputs_of_GIS_Preprocessor.py` script takes the output ZIP file generated above and creates a raster from each 2D variable for examining the result.

The tool will create the output folder if it does not already exist, and write all results to that location.

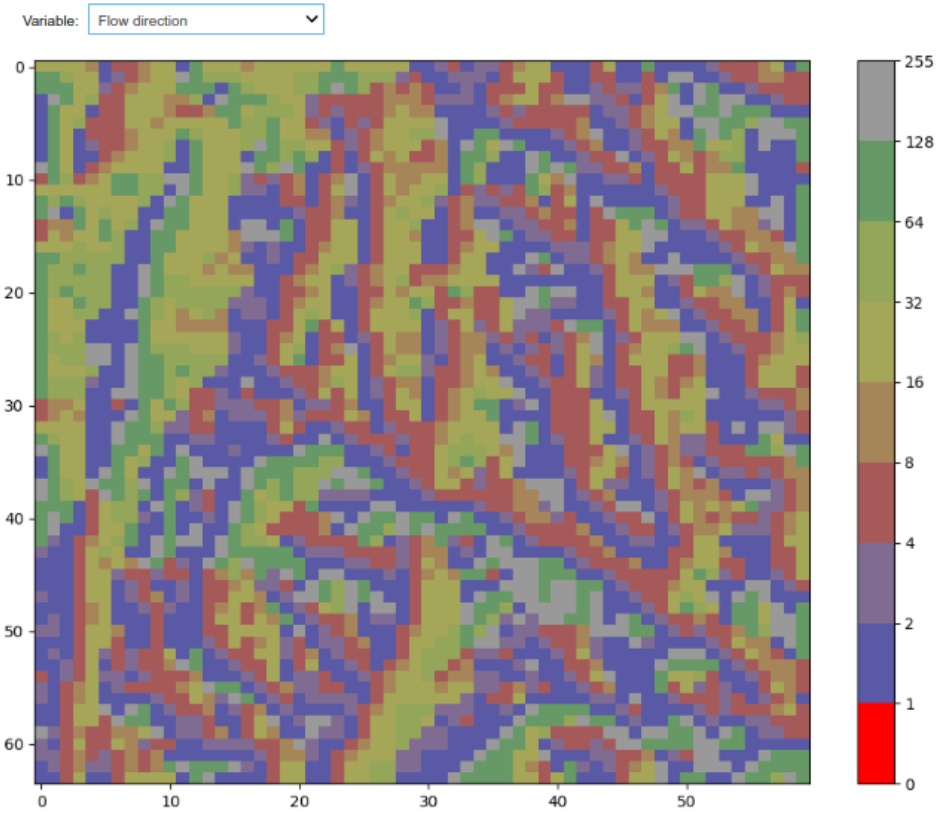
```
import ipywidgets as widgets
from ipywidgets import interact

def see_raster(x):
    src = rasterio.open(os.path.join(raster_outputs, f"{x}.tif"))
    cmap, norm = cmap_options(x)
    if x in ['TOPOGRAPHY']:
        pyplot.imshow(src.read(1), cmap=cmap,  aspect='auto', norm=n
        orm, interpolation='nearest', vmin=0)
    else:
        pyplot.imshow(src.read(1), cmap=cmap,  aspect='auto', norm=n
        orm, interpolation='nearest')
    cbar = pyplot.colorbar()

    # Keep the automatic aspect while scaling the image up in size
    fig = pyplot.gcf()
    w, h = fig.get_size_inches()
    fig.set_size_inches(w * 1.75, h * 1.75)

    # Show image
    pyplot.show()
```

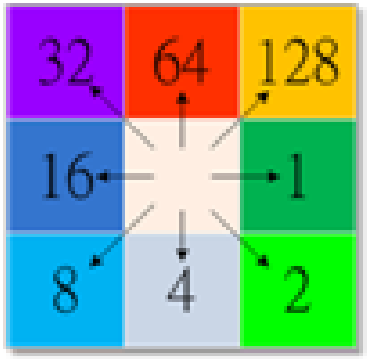
```
in_raster = widgets.Dropdown(
    options=[('Basin', 'BASIN'), ('Basin mask', 'basn_msk'), ('Chann
        el grid', 'CHANNELGRID'), ('Flow accumulation', 'FLOWACC'),
            ('Flow direction', 'FLOWDIRECTION'), ('Forecast points',
        'frxst_pts'), ('Lake grid', 'LAKEGRID'),
            ('Land use', 'landuse'), ('Latitude', 'LATITUDE'), ('LKS
        ATFAC', 'LKSATFAC'), ('Longitude', 'LONGITUDE'),
            ('OVROUGHRTFAC', 'OVROUGHRTFAC'), ('RETDEPRTFAC', 'RETDE
        PRTFAC'), ('Stream order', 'STREAMORDER'),
            ('Topography', 'TOPOGRAPHY')],
    value='FLOWACC',
    description='Variable:')

interact(see_raster, x=in_raster)
```

Elevation



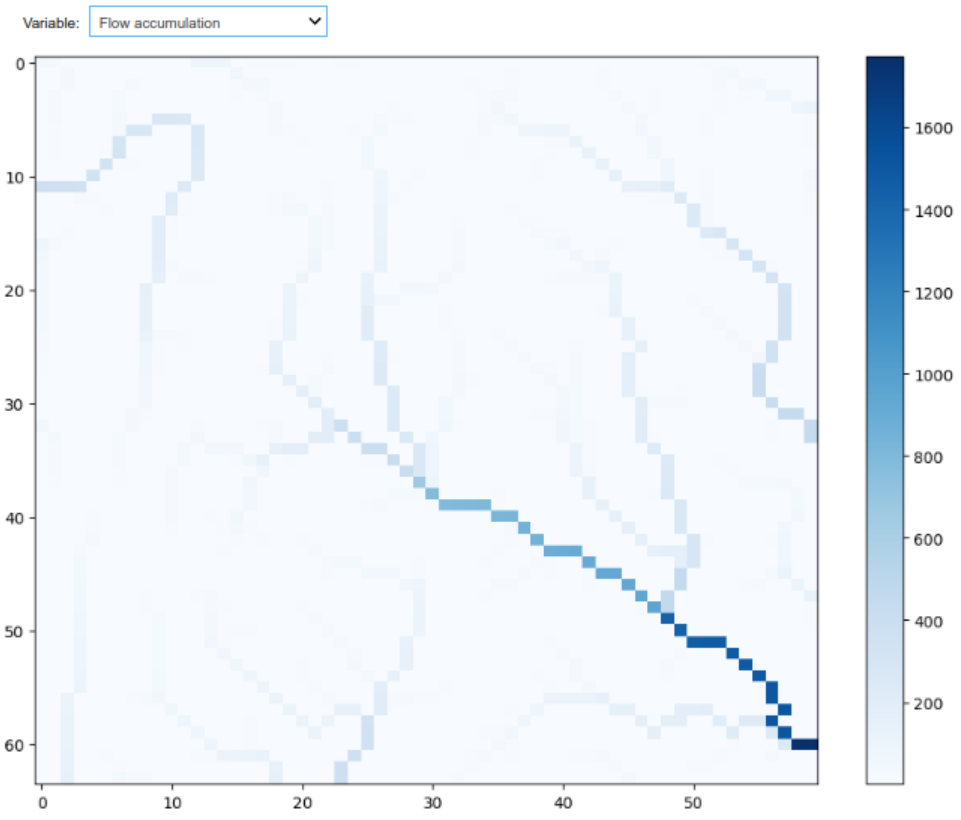
Flow Direction





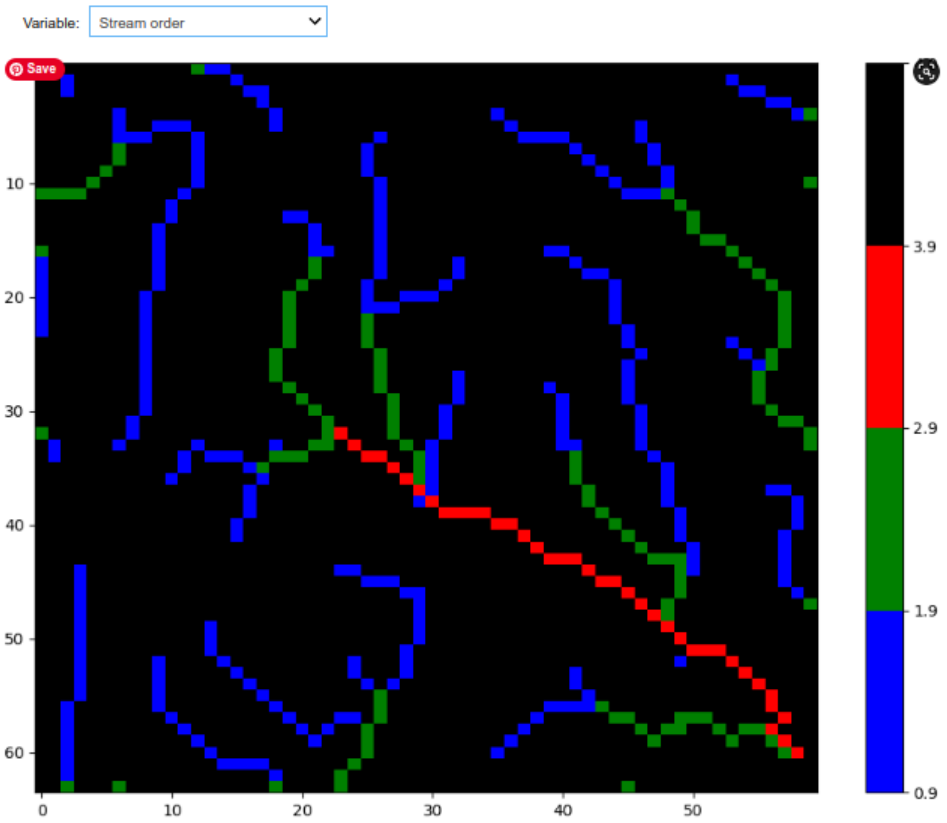
Direction Coding

Flow Accumulation

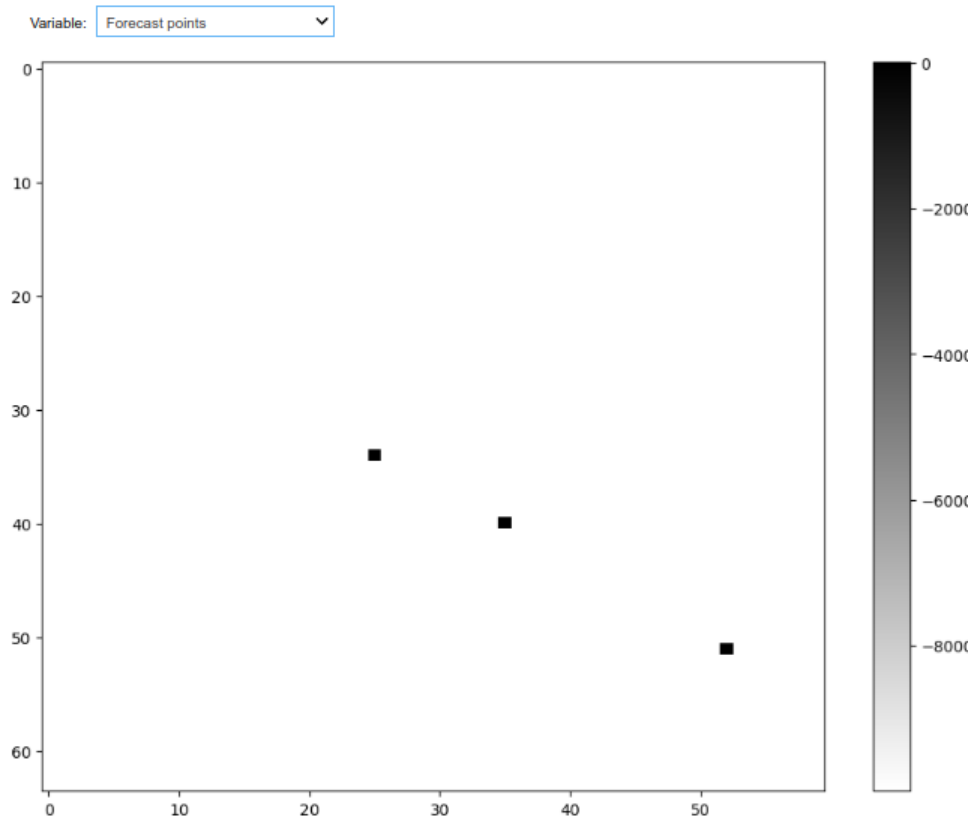


Channel Grids

Stream order grid



Gage locations

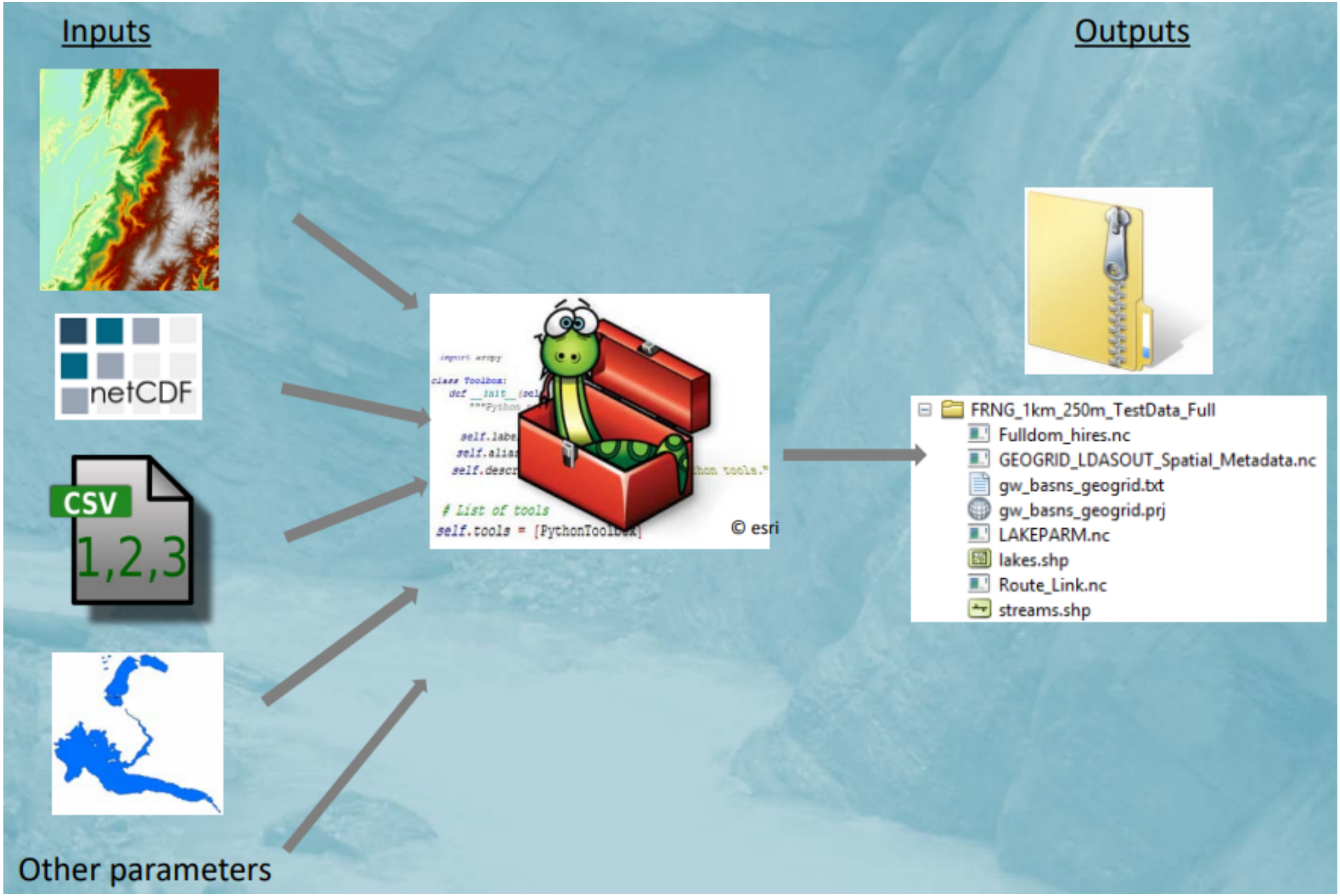


More information on NCAR WRF-Hydro GIS Preprocessor Python Scripts (https://github.com/NCAR/wrf_hydro_gis_preprocessor (https://github.com/NCAR/wrf_hydro_gis_preprocessor))

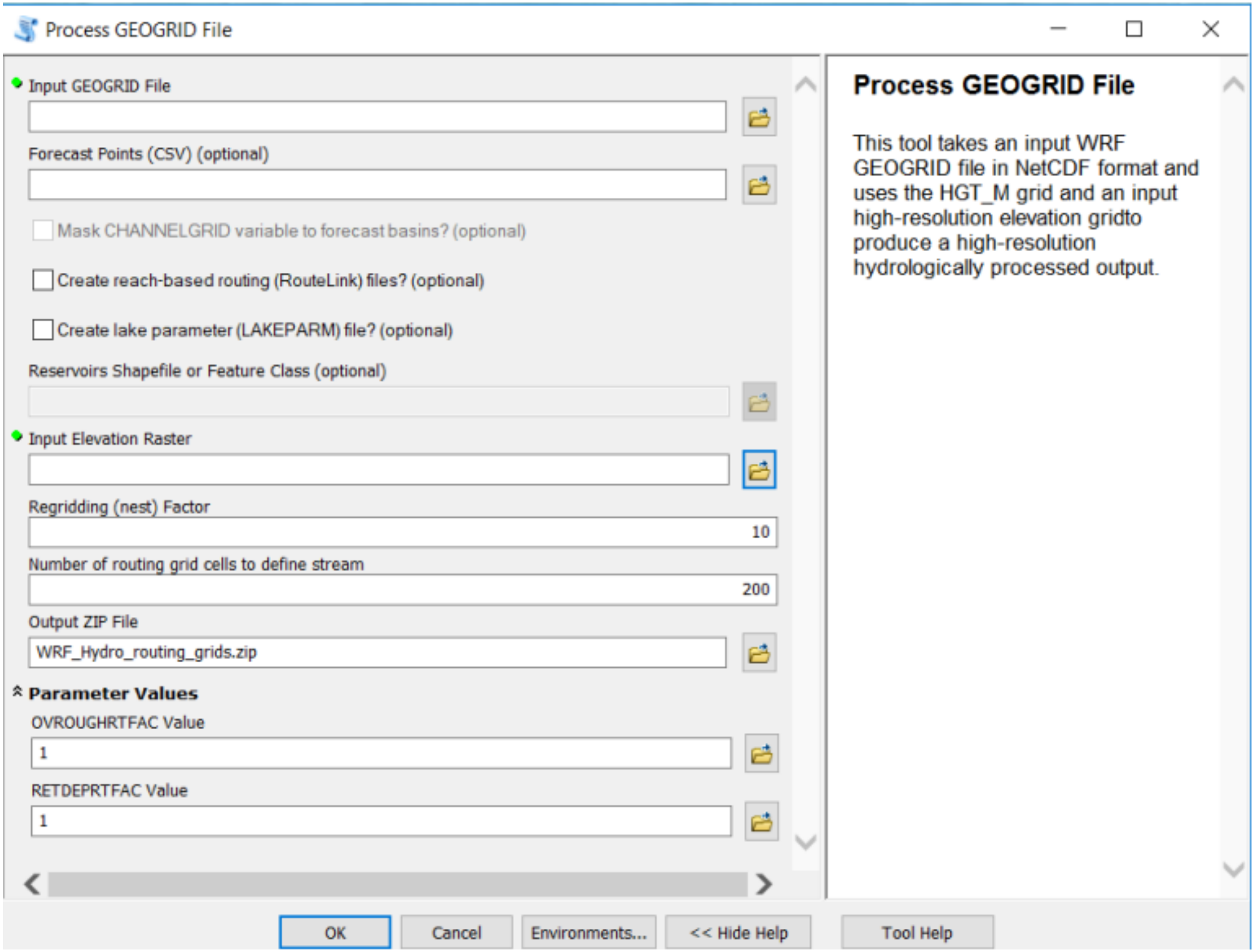
1.2 Method 2: WRF-Hydro ArcGIS Pre-processing Tools

- written in Python, using ArcGIS python API (arcpy)
- ArcGIS for Desktop (Version 10.3.1+)
- DO NOT USE ArcGIS 10.4, 10.5
- Basic, Standard, or Advanced license levels
- Spatial Analyst extension required

1.2.1 ArcGIS Pre-processing Tool workflow:



1.2.2 Preview of ArcGIS Pre-processing Toolbox


Process GEOGRID File
Input GEOGRID File
Forecast Points (CSV) (optional)
Mask CHANNELGRID variable to forecast basins? (optional)
Create reach-based routing (RouteLink) files? (optional)
Create lake parameter (LAKEPARM) file? (optional)
Reservoirs Shapefile or Feature Class (optional)
Input Elevation Raster
Regridding (nest) Factor
10
Number of routing grid cells to define stream
200
Output ZIP File
WRF_Hydro_routing_grids.zip
Parameter Values
OVROUGHRTFAC Value
1
RETDEPRTFAC Value
1
Process GEOGRID File
This tool takes an input WRF GEOGRID file in NetCDF format and uses the HGT_M grid and an input high-resolution elevation gridto produce a high-resolution hydrologically processed output.
OK
Cancel
Environments...
<< Hide Help
Tool Help

Link: https://github.com/NCAR/wrf_hydro_arcgis_preprocessor (https://github.com/NCAR/wrf_hydro_arcgis_preprocessor)