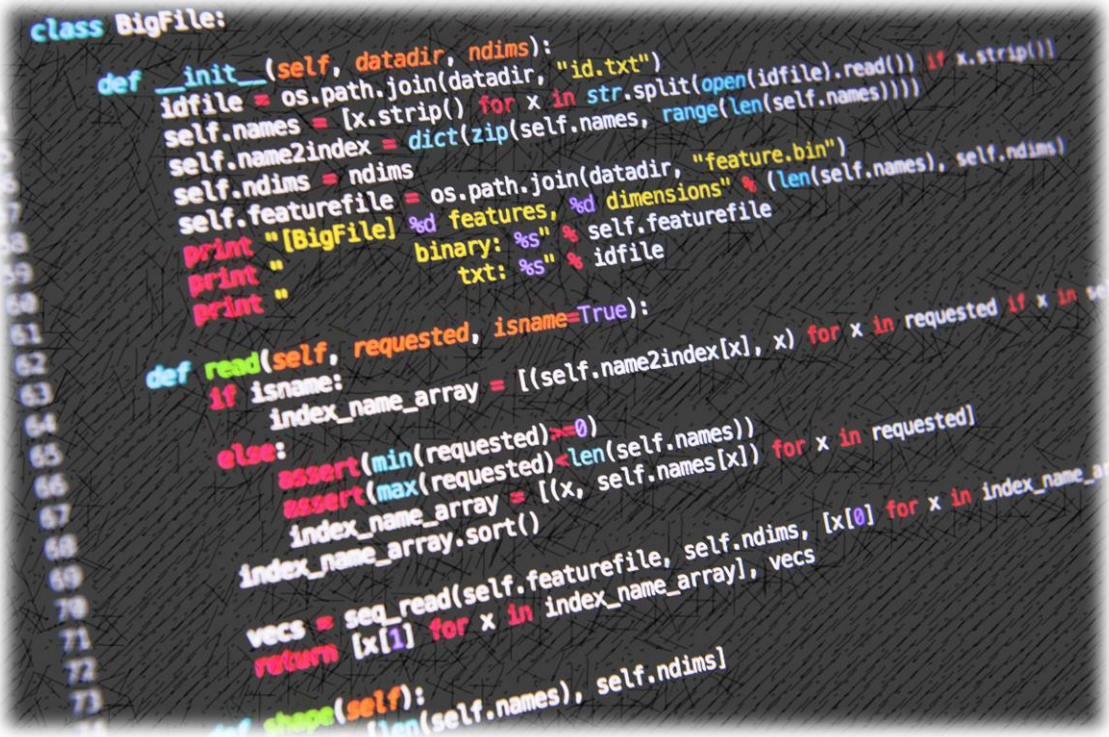
Flooding Risk Programming



David Merino Ramos

EGM722

Programming for GIS and Remote Sensing

Introduction

The aim of this code is to generate vector data representing affected area by flooding, based on a Digital Terrain Model (DTM). By specifying the water flood level (real or expected) the code identifies those areas bellow this level and transform them into polygons, allowing further spatial analysis, and the visualisation of affected areas.

Set up/installation

The link to the repository is: https://github.com/Polymerizable/project

There you can find a public folder called “project” with various files.
“Contour_polygons_from_DTM.py” python file is the code that you have to run.

To run the code, follow the next steps:

1. Set up Anaconda:
   - Download and install Anaconda from the official website (“https://www.anaconda.com/download”)
   - Open Anaconda Navigator and create a new environment for your project.
   - Install Python and necessary packages in the environment using Anaconda Navigator or the command line.
2. Set up PyCharm:
   - Download and install PyCharm from the official website (“https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows&code=PCC”).
   - Open PyCharm and create a new project in the desired directory.
   - Configure the project interpreter to use the Anaconda environment created (see environment_project.ylm).
   - Create a new Python file in your project directory and paste the code into it.
   - Run the code in Pycharm and check the results into the specified folder.

Methods

The general idea of how works the code is:

- ✓ The input consists of raster files (in "file.tif" format) representing tiles of the DTM.
- ✓ The transformation involves applying a mask to the raster to select only those locations that are below the flood level, and transforming them into polygons representing the affected areas.
- ✓ The output consists of shapefiles representing these polygons for each tile.

The code performs the following steps:

1. Define the Coordinate Reference System (CRS) as British National Grid (EPSG:27700).
   As the project test data is from England the CRS selected was the British National grid, but it is possible to work with others if you specified it.
2. Prompt the user to input the expected water flood level in meters.
   As the DTM has float values, the flood level should be floating-point to allow future comparations between values. There is a small script that ensures that the introduce value is from a valid type.
   An infinite loop is created to ask the user the flood level. If it possible to transform into float type, we break the loop, if not ask for a new input.

```
while True:
    #  Ask for the water flood expected level.
    flood_level = input("Please enter the expected water flood level (meters):")
    # Check if the input is a valid floating-point number
    try:
        flood_level_input = float(flood_level)
        break
    except ValueError:
        print("Invalid input. Please enter a valid floating-point number.")
```

Figure 1 Asking for a floating-point value of the expected flood level.

3. Create a list with all files located in folder and iterate through each item.

It retrieves a list of all files and directories in the specified directory (`folder_path`) using the `os` module. It aims to allow iterate through each file in this list.

```
# Define folder path. The second option is to ask for it
folder_path = 'D:/.../Project_Data/DTM_tif'
#  folder_path = input("Please introduce the folder path that contain the data:")
```

Figure 2 Setting the folder path.

```
# Create a list with all the files in the folder
files = os.listdir(folder_path)

# Iterate through each file in the folder to work with them
for file_name in files:
```

Figure 3 Files list creation and loop for iteration with each item of the list.

4. Iterate through each GeoTIFF file in the specified folder.
   Separate the name of the file between name and extension. If statement that run the process if True value is found.

```
# Iterate through each file in the folder to work with them
for file_name in files:
    # Split the file_name into base_name and extension
    base_name, extension = os.path.splitext(file_name)

    if extension == '.tif':
        # Define the path of the file
        file_path = os.path.normpath(os.path.join(folder_path, file_name))
```

Figure 4 Selecting procedure for '.tif' files

5. Read the raster data as a NumPy array using rasterio.

```
with rio.open(file_path) as dataset:
    # Read the raster data as a numpy array
    data = dataset.read()
```

Figure 5 Use of rasterio module to read the data

It opens the raster file located at 'file_path' using 'rasterio' (as 'rio') as a dataset. The 'with' statement is used for ensuring that the file is properly closed after accessing it. Data is a NumPy array that contains the raster data loaded from the file in a numerical format.

6. Create a mask to select pixels with values less than or equal to the specified flood level.

```
# Create a mask to select pixels with values higher than the threshold
mask = data <= flood_level_input

mask = np.where(mask, 1, 0)  # Convert True to 1, False to 0
```

Figure 6 Mask creating to select only affected pixels with a boolean output.

The output of the `mask` variable will be a NumPy boolean array that represents the result of the comparison. Later it is transformed into a NumPy binary array.

7. Generate shapes from the mask using rasterio.features.shapes().

```
# Generate shapes from the mask
shapes_generated = rasterio.features.shapes(mask, transform=dataset.transform)

# Convert shapes to polygons
polygons = [shape(geom) for geom, value in shapes_generated if value == 1]

# Create a GeoDataFrame from the polygons
gdf = gpd.GeoDataFrame(geometry=polygons, crs=dataset.crs)
```

Figure 7 Code that includes the data selection from the raster and its transformation into polygons.

The variable 'shapes_generated' will hold a generator object containing tuples where each tuple represents a shape (polygon) along with its associated value (0,1) from the mask using.

8. Convert the generated shapes to polygons using shapely.geometry.shape(). The output of this would be a list of polygons created from the shapes associated with a value of '1' (meets criteria). Each element in the `polygons` list would represent a Shapely polygon object based on the corresponding geometry ('geom') extracted from the shapes (created in memory, they have a

memory address represents the location in memory where each 'Polygon' object is stored).

9. Create a GeoDataFrame from the polygons, specifying the CRS obtained from the raster dataset.

   It creates a GeoDataFrame object, which consists on a tabular data structure that extends the capabilities of pandas DataFrames to incorporate spatial data like points, lines, and polygons, in this case it includes a 'geometry' column containing the spatial information with the polygons stored in it.

10. Save the GeoDataFrame to a shapefile.

```
# Save the GeoDataFrame to a shapefile
output_shapefile = os.path.normpath(os.path.join(folder_path, base_name + '.shp'))
gdf.to_file(output_shapefile)
```

Figure 8 Creation of the shapefile.

It is possible to get data from the EarthExplorer (EE), which is a web application that allows users to search, discover, and order satellite, aircraft, and other remote sensing data from the USGS.website (https://earthexplorer.usgs.gov/). NB: registration is both free and required.

- Once you are logged select an area and create a polygon by clicking on the map.

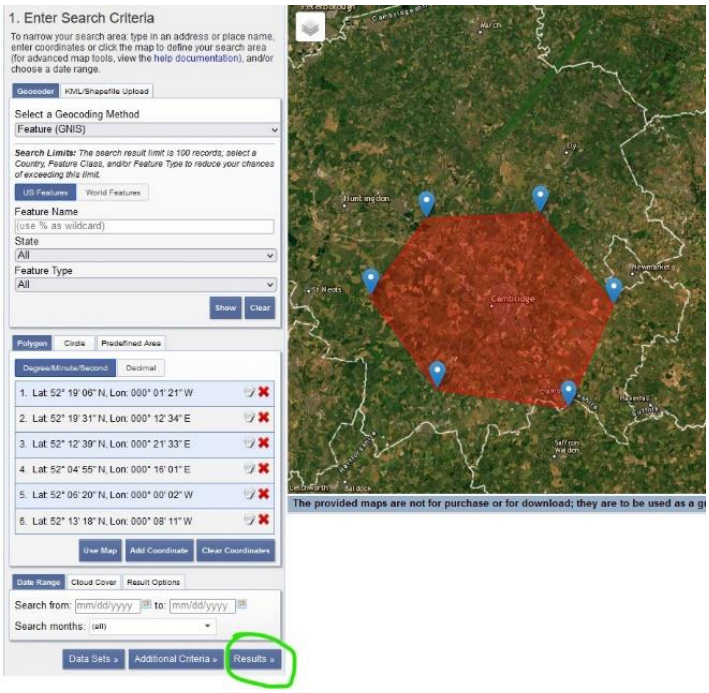


Figure 9 Map selection in Earth Explorer (USGS).

- Second step: Select the data set. In the menu open digital elevation and SRTM. Tick STRM 1 Aro-Second Global (30x30m raster)

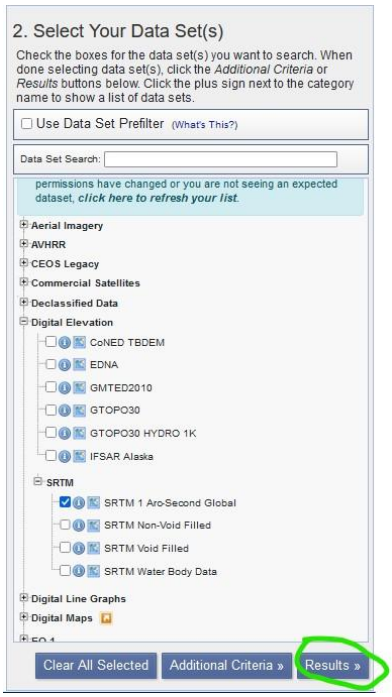


Figure 10 Digital elevation data set selection.

- View the results and select those you need. Click on the download icon.

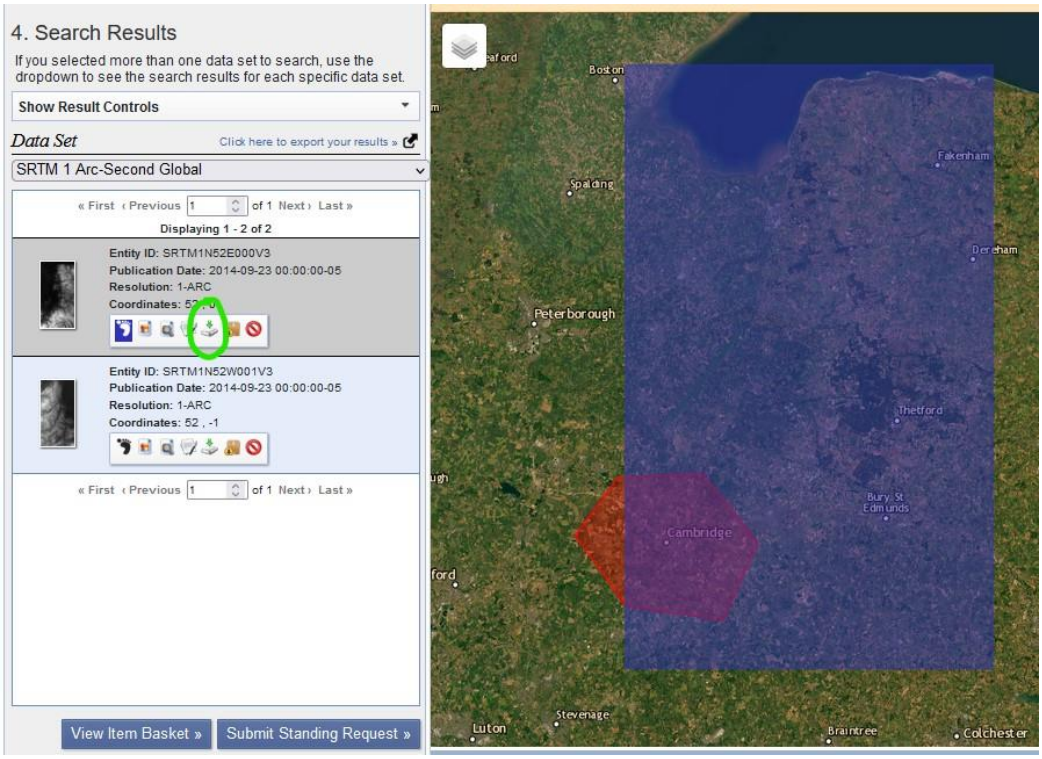


Figure 11 Disponible data set.

- Choose GeoTIFF format to download.

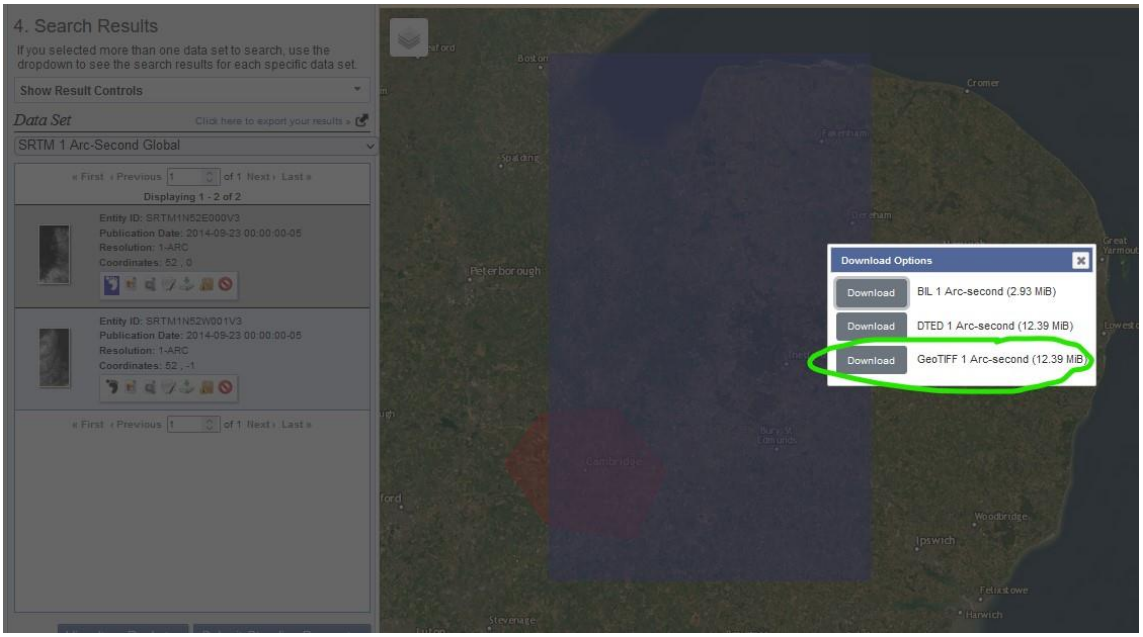


Figure 12 Select and download GeoTIFF data.

Future work directions

A good strand of work would be to select only those polygons that intersect with the river course, ensuring that those are the actual affected areas. Another aspect to consider would be the merging of all affected polygons that are adjacent to each other.

Results

This script will create a shapefile per each '.tif' file present in the folder. This set of shapefiles represent the area affected by flooding. The shapefiles will contain polygons representing flooded areas based on a stipulated flood level.

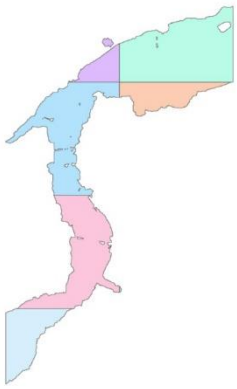


Figure 13 Polygons test output. They follow the river shape.

The following aspects may be noted:

1. There is any check that verifies that the '.tif' files represent a DTM with height values.
2. The Vertical Datum is not specified, so the flood level (meters) should be considered in the same way as the Height values of the raster.
3. As each tile of the DTM is a separate file, the result of the flooded area will be composed of different adjacent polygons.
4. If the area analysed is completely above the flood level a warning message will appear: "UserWarning: You are attempting to write an empty DataFrame to file. For some drivers, this operation may fail. _to_file_fiona(df, filename, driver, schema, crs, mode, **kwargs)" as the script generates an empty file. It has been tested with

any further issue than the warning. Note that it can be improved avoiding saving empty dataframes.

Troubleshooting tips

- Ensure all dependencies are correctly installed.
- Check that the specified folder path contains valid GeoTIFF files with the .tif extension.
- Verify that the input flood level is a valid floating-point number.
- Check that the CRS is correctly defined and compatible with the input raster data.