FAIR SCIENCE – Data Access Module #

Downloading Digital Elevation Model from National Elevation Dataset

Prepared by Sayan Dey and Venkatesh Merwade
Lyles School of Civil Engineering, Purdue University
vmerwade@purdue.edu

1. INTRODUCTION

Digital Elevation Models (DEMs) are the most common representation of topography in hydrologic and hydrodynamic applications. The United States Geological Survey (USGS) has created the National Elevation Dataset (NED), which provides seamless topographic representation of the entire Continental United States (CONUS). The NED is available for download in the form of tiles of size 1° × 1° (latitude-longitude).

The objective of this exercise is to learn to access and download NED tiles overlapping a study area programmatically. This may involve the automated download of multiple NED tiles, depending on the size and location of the study area. Students are expected to have a basic understanding of GIS file formats (raster, shapefile) and Python.

2. COMPUTER REQUIREMENTS

You must have a web browser, connection to the internet and login credentials for an account on www.mygeohub.org.

3. DATA REQUIREMENTS

The key data required for this exercise is a polygon shapefile depicting the region of interest (study area) for which the DEM is to be downloaded. You can download an example shapefile from the following link for this exercise: <link>. **Download** the zipfile from the link into your local drive and unzip it. Upload the shapefile to a folder in mygeohub. This folder will henceforth be referred to as input folder.

The example shapefile is a sub-watershed of the Wabash River, with its outlet at West Lafayette, IN. You can use any polygon shapefile for this exercise provided it only contains one polygon feature and the polygon feature is a simple polygon.

You are also provided with a jupyter notebook, *init_NED_code.ipynb,* for this exercise. It has code for initializing PyQGIS, and some user defined functions that you will be using to download the NED tiles. The main code, where you call the user-defined function, is incomplete and you will be guided by this tutorial to complete the code and programmatically download the NED files.

4. GETTING STARTED

Working with geospatial data requires the use of geographic information systems (GIS). Here we use QGIS, an open source and free GIS software. Python support is available for QGIS

through its Python QGIS API called PyQGIS (if you are interested in some additional reading, visit: https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/). PyQGIS is already installed in the mygeohub Jupyter notebook.

4.1 Initialize PyQGIS

Open a web browser and login to www.mygeohub.org using your credentials. **Click** on *Resources >> Tools >> Jupyter Notebook with Anaconda 5.1 >> Launch Tool*. **Click** on the blue ribbon to go to your home directory. **Navigate** to your work folder and **click** on *New >> Jupyter Notebook.* **Rename** your jupyter notebook to "NED_Download" by clicking on "Untitled" near the top.

Make sure the kernel is set to [conda: qgis] by clicking on *Tools >> kernel >> [conda:qgis]*. The first task is to initialize PyQGIS which is established with the following code. Copy the first cell from the provided jupyter notebook (*init_NED_code.ipynb)* code.

```
# INITIALIZE PYQGIS (run once per session)
import sys, os
os.environ['QT_QPA_PLATFORM']='offscreen'
sys.path.append('/apps/share64/debian7/anaconda/anaconda3-5.1/envs/qgis/')
from qgis.core import *
from qgis.analysis import QgsNativeAlgorithms
#from qgis.utils import *
import processing
from processing.core.Processing import Processing

qgs = QgsApplication([], False)
qgs.initQgis()
Processing.initialize()
QgsApplication.processingRegistry().addProvider(QgsNativeAlgorithms())
```

Figure 1: Initializing PyQGIS in Jupyter notebook

Note: You only need to run this cell once each session. Running this cell again (when PyQGIS is already initialized) may crash the kernel. If that happens, restart the kernel, clear all outputs and then run this cell once.

4.2 Import libraries and provide inputs

The next step is to import libraries (modules) that will be used for carrying out specific tasks throughout the notebook. Import the four libraries as shown below in a new cell.

```
# IMPORT LIBRARIES
import ftplib, math, zipfile, shutil
```

Figure 2: Importing libraries (modules)

The ftplib module is used for connecting to ftp sites and searching and accessing their contents. The math module provides access to simple math functions such as sum, mean, floor among

others. The zipfile module is used for zipping and unzipping zip files. Finally, shutil module contains functions for operations on files, folders or collection of files.

Note that the sys and os modules have already been imported during PyQGIS initialization in Figure 1.

Next, we need to provide the program with the location of *input* and *output folders* and the name of the input shapefile (boundary_file) as shown in Figure 3 below. The input folder is the one which has the shapefile. A *work folder* is created inside the input folder and all intermediate files are stored in the work folder. The output folder is the one where the final output will be stored (this folder is part of the Data Processing Module and can be ignored for this module).

Make sure to include .shp in the boundary file name. The input folder name follows the syntax: /home/mygeohub/<mygeohub id>/<path to the folder where shapefile is located>.

```
# USER INPUTS
input_folder_name = "/home/mygeohub/joseph57/EAPS_591/HydroModule01/Input1"
output_folder_name = "/home/mygeohub/joseph57/EAPS_591/HydroModule01/Output1"
boundary_file =  "watershe_prj.shp"
```

Figure 3: Name of input folder, output folder and boundary file.

4.3 User-defined functions

We define two user-defined functions: one for unzipping zipped files (*UnzipNED*) and one for downloading NED (*DownloadNED*) tiles. These are provided in *init_NED_code.ipynb*.

They are explained below:

UnzipNED: This user-defined function (Figure 4) takes a zipped filename as input and unzips it. The zipped file must be located in the *work folder*. The unzipped files are also extracted in the *work folder*.

```
# USER DEFINED FUNCTION FOR UNZIPPING DOWNLOADED TILES
def UnzipNED(f):
    try:
        zfile1 = zipfile.ZipFile(work_folder_name + "/" + f, 'r')
        zfile1.extractall(work_folder_name)
        zfile1.close()
        print("NED unzipped successfully: " + f)
    except:
        print("Error in unzipping: " + f)
```

Figure 4: User-defined function for unzipping zipped files.

DownloadNED: This user-defined function takes in as input two integers – latitude and longitude – and downloads the NED tile corresponding to that latitude and longitude. The NED tiles have the string "n<lat>w<long>" in their name.

A NED tile with the name string "nYYwXX" cover a 1° × 1° spatial extent spanning (YY-1) to YY and (XX-1) to XX. For example, a tile with the name string "n41w87" cover the region between latitudes 40°N to 41°N and 86°W to 87°W.

The function first connects to the NED ftp site (rockyftp.cr.usgs.gov) and then navigates to the folder where the zipped NED tiles are stored in the ftp server. In that folder, it searches and downloads the zipped file with the string "n<lat>w<long>" in its name. In case there are more than one zipped file with the same string in its name, it downloads the one that is largest in size.

```
# USER DEFINED FUNCTION FOR DOWNLOADING TILES
def DownloadNED(lat,lon):
    name = "n"+lat+"w"+lon
    with ftplib.FTP('rockyftp.cr.usgs.gov') as ftp:
        try:
            ftp.login()
            ftp.cwd('vdelivery/Datasets/Staged/Elevation/1/ArcGrid/')
            contents = ftp.nlst()
            filtered_contents = [f for f in contents if ((name in f) & (".zip" in f))]
            if len(filtered_contents) == 0:
                print("No file found for: " + name)
            elif len(filtered_contents) == 1:
                final_file = filtered_contents[0]
                print("1 file found for current tile")
                print("Downloading " + final_file)
                fo = open(os.path.join(work_folder_name, final_file), 'wb')
                ftp.retrbinary("RETR " + final_file , fo.write)
                fo.close()
                print("Download successful")
                return(final_file)
            elif len(filtered_contents) > 1:
                print("More than 1 file found: Dowloading largest zip file")
                file_list = []
                ftp.sendcmd("TYPE i")
                for f in filtered_contents:
                    file_list.append((f,ftp.size(f)))
                file_list.sort(key=lambda s: s[1])
                final_file = file_list[-1] #return the largest file
                print("Downloading..." + final_file)
                fo = open(os.path.join(work_folder_name, final_file), 'wb')
                ftp.retrbinary("RETR " + final_file , fo.write)
                fo.close()
                print("Download successful")
                return(final_file)
            else:
                print("Unknown error with file download for:" + name)

        except ftplib.all_errors as e:
            print('FTP error:', e)
```

Figure 5: User-defined function for downloading NED files.

4.3 Main code: Putting it all together

We now arrive at the final step, which is to use these user-defined functions to download all the NED tiles that cover our study area.

First, we find the extents of the boundary polygon, that is, we find the northmost and southmost latitudes and the eastmost and westmost longitudes that enclose the polygon. To do this, we convert the coordinate system of the boundary polygon to NAD1983 coordinate system and then use the extent property of a polygon object in QGIS to extract the bounds. Print these extents as shown in Figure 6.

Exercise 1: Can you use the extents to create a list of latitudes and longitudes for the relevant NED tiles to be downloaded for the input polygon?

Hint: Taking the ceiling or the floor (depending on the location) provides the integer latitude and longitude that make the bounds completely containing the input boundary polygon. Use the range function to generate all intermediate latitude and longitudes.

Once we find all the $1° \times 1°$ NED tiles that overlap the bounds of the boundary polygon by looping across all integer latitudes and longitudes contained in the bounds. For each set of integer latitude and longitude, we get one NED tile. The latitude-longitude pairs are fed one by one to the *DownloadNED* function which downloads the relevant zipped NED tile. The output of *DownloadNED* is fed into *UnzipNED* which then unzips these files.

Exercise 2: Create for-loops to go through the list of latitudes and longitudes created in Exercise 1 and call the *DownloadNED* and *UnzipNED* functions at appropriate places in the code for downloading all NED tiles overlapping with the input polygon. Provide appropriate print statements and comments to keep track of the progress of downloads.

After running the code, go to your input folder. You will find that a work folder has been created inside your input folder. Inside the work folder, you will find all the tiles (both zipped and unzipped) that overlap the input polygon boundary.

These tiles need to be preprocessed into a single DEM with the same coordinate system as the input boundary polygon before they can be used in any application, which we will do in the Data Processing Modules.

Ok, you are done… for now!!!