Thru-hike terrain analysis: Is the PCT, CDT, or AT better?

March 19, 2020

Micah Goldade

Abstract

This analysis compares the terrain along three major long-distance thru-hike trails in the United States to identify areas of geological complexity. The terrain is characterized based on Digital Elevation Model (DEM) data derived from the Shuttle Radar Topography Mission (SRTM) dataset. This analysis attempts to derive an appropriate frequency signature for sections of the trail using Fast Fourier Transform (FFT) on elevation profiles filtered by the Continuous Wavelet Transform (CWT). These signatures are then analyzed using Singular Value Decomposition (SVD) to identify the relative complexity of the terrain on these trails.

1 Introduction and Overview

The United States is famous for its large variety of geographic regions, particularly those protected by organizations including the National Parks System. One of the more extreme and romanticized methods to enjoy these preserved areas is to plan a thru-hike of one of the three major long-distance hiking trails - the Pacific Crest Trail (PCT), the Continental Divide Trail (CDT), or the Appalachian Trail (AT).

Attempting a thru-hike of any one of these trails can be a years long effort for planning and execution. As of 2018 less than 400 hikers have completed all three and been awarded the Triple Crown of Hiking.

Fig 1. The three primary long-distance thru-hikes in the United States

Since seeing all of this geography in person is out of reach for most of us, I wanted to perform an analysis on the terrain along these trails to try to **1)** determine a frequency signature that helped describe the topology on a given point along the trails, **2)** determine if any portions of the trails have a particularly unique terrain frequency signature, and **3)** determine which trail has the most complex terrain profile and is best (*just kidding, I'm sure they are all great*).

2 Theoretical Background

The largest challenge for this project was researching, understanding, locating, and transforming geographic terrain datasets suitable for this analysis.

At first, it seems like the process should be relatively simple. Large mapping projects such as Google Maps and OpenStreetMap have synthesized large collections of terrain data, showing it is possible. Furthermore, much of the data required for this analysis was procured by the United States Geographical Survey (USGS), and therefore should be free and easy to download and use. However, this over-simplification misrepresents how complex the process of generating terrain data to generate Geographic Information System (GIS) data can be. As a naive engineer with no previous GIS experience, I quickly learned just how complicated the process can be, especially when needing to work with minimally processed source data. The following are the main concepts I needed to work through to complete the analysis.

2.1 Digital Elevation Models

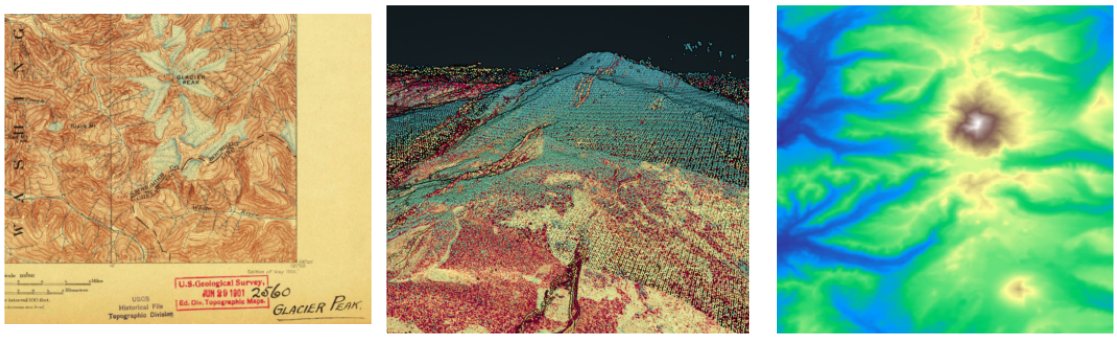
The primary terrain of interest for this analysis is the elevation profile. Attempting to use data for other terrain features, such as vegetation, would be both difficult to find, and also difficult for

comparison since not all areas have vegetation. However, all regions will have an elevation profile.

Elevation data is typically processed as a Digital Elevation Model (DEM). Although there are vector based DEMs, the most common DEM type appears to be as a raster, with a 2 dimensional matrix of equally spaced x-y coordinates containing elevation. This DEM raster data format is well-suited for this analysis, and will be the primary data format.

An important consideration of DEM raster data is what dataset it was based on and how it was processed, since there is an large variation in both quality and availability of elevation data.

Fig 2. Example of digital elevation data sources for Glacier Peak, WA
Historical survey maps (left); High resolution Lidar data poitn cloud (middle); Radar data processed into DEM raster, used in this analysis (right)



2.1.1 Manual land surveying

The original data source for DEM maps came from manual land surveys. The DEM datasets relying on this data are based on historical topographic maps. Although they have been a great resource, these datasets are being enhanced to higher resolutions using standardized data collection methods noted below.

2.1.2 Photogrammetry

Photogrammetry finds correlations in photographed images to build a DEM dataset. This process can add accuracy to land surveying methods. Some of the limitations of this dataset include attempting to create a "bare-earth" model without vegetation, since areas like thick forests are hard to process.

2.1.3 Lidar

Lidar, which illuminates a target with light and measures reflection, can create an even more accurate DEM model, especially in areas with vegetation since the illumination methods can penetrate some foliage.

Lidar datasets are often the highest resolution datasets available, and this method can also create some incredible looking 3D point-cloud datasets. These point cloud sets are then processed to generate highly DEM raster dataset for analysis.

Unfortunately these datasets are less reliable for this type of analysis, since the availability of consistent data is not yet there. Some regions have great coverage of Lidar data, but most areas of the US, and most of the trail systems, do not have any Lidar data available.

2.1.4 Radar

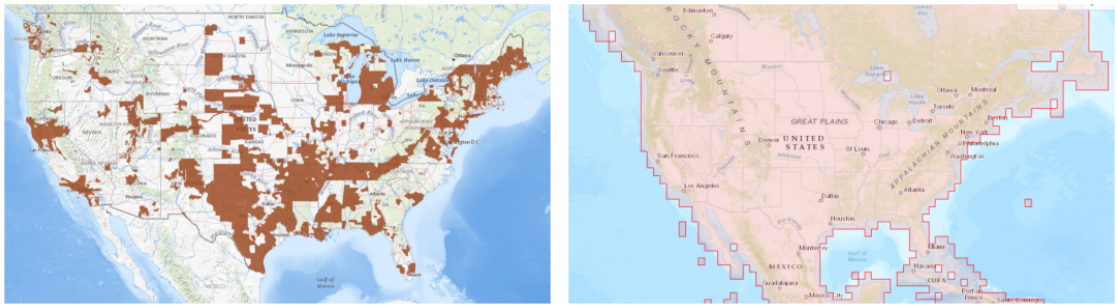
Most useful dataset for this type of analysis is obtained by radar missions, such as the Shuttle Radar Topography Mission (SRTM), which used radar to obtain consistent elevation data on a near global scale. Although the resolution is significantly lower for radar than that of some Lidar datasets (e.g. 30 meter resolution vs 3 meter resolution), the consistency and scope of the STRM dataset made it by far the most useful type of dataset considered. The lower resolution is also not a concern, since we will focus the analysis on larger features anyways due to computational resource limitations.

> Note: Although Lidar and Radar datasets are often available with "raw" source of x-y-z coordinates, the most useful format to work with in this analysis is a DEM raster file. These DEM files have been filtered and processed, potentially enhanced by information from additional data sources, and this analysis assumes the USGS will do a much better job of filtering the raw data than I ever could.

2.2 Data Sources

The primary source of elevation datasets for the United States is the USGS, which obtains, process and maintains the majority of the data. They are a great resource to find and review specific dataset, but their web tools can be a challenge to work with in obtaining consistent data.

Fig 3. Example of available Digital Elevation Model (DEM) data resolutions in the US
1 meter resolution on left; 30 meter resolution on right



2.2.1 The National Map

The National Map is an awesome project intended to combine a large amount of mapping information into one data access portal. It is useful for exploring which data is available beyond just elevation data. It is also simple to download individual datasets, including raw Lidar data. However, it is limited in the ability to stitch together DEM files.

2.2.2 OpenTopography

Fortunately, the OpenTopology organization has setup a great method to pull elevation datasets from various sources, including STRM. In particular, they have an RESTful API leveraged in the analysis to pull consistent STRM data stitched into the sizes needed. This saves an extremely large amount of tedious data transformation and processing.

> Note: OpenTopography also has access to higher resolution global datasets. However, it appears these higher resolution sets change resolution depending on what is available within a given region. Therefore, for a consistent comparison, we use the STRM data directly.

2.3 Geographic Projections and Coordinate Reference Systems

I could boil the majority of the complexities with this analysis to one issue - the earth is unfortunately not flat. Furthermore, it isn't even a very good sphere. This complexity is abstracted away when mapping products are produced for specific use cases. However, when combining datasets from various sources using an appropriate coordinate transformation for the datasets is of the utmost importance.

There are thousands of Coordinate Reference Systems (CRS) maintained in the European Petroleum Survey Group (EPSG) Geodetic Parameter Dataset registry, now maintained by the International Assosication of Oil and Gas Producers (IOGP) Geomatics Committee. The datasets available for trail maps and topography in general may be provided in any one of the various coordinate systems. It is therefore important to transform the data to the coordinate systems most appropriate for the analysis.

We will focus on two coordinate systems specificed in the EPSG - the World Geodetic System and the Web Mercator Projection.

2.3.1 World Geodetic System

The World Geodetic System (WGS) is used when accuracy is required. The coordinate includes an approximation of the Earth's shape in an Elliptic coordinate system (these shape approximations are known as geodetic reference datums). Although more accurate geodetic datums exist based on more complex shapes, the WGS has become the primary standard for global accuracy, is accurate up to a couple meters, and used for satellite navigation including GPS.

The WGS data format is the standard used for the SRTM dataset, and is used as an input to the OpenTopology API. Note that when calculating distances in WGS format, the conversion approximation of 1 degree = 111.320 km will be used. This allow the convenience of a consistent shape of the DEM files with the minor sacrifice of a slightly less exact distance measurement.

> Note: WGS is specified as **EPSG:4326**.

2.3.2 Web Mercator Projection

The generally population views and works with data on flat maps, particularly through using web mapping applications such as Google Maps and OpenStreetMaps. These maps use the web-Mercator Projection, and therefore for visualization we will use the same standard. This projection is actually based directly on WGS, but sacrifices accuracy for ease of use (e.g. the units are in meters from the Universal Transverse Mercator (UTM) coordinate system, and not degrees). Therefore, it is not recommended to use for calculations, and only to use for visualization.

Note: The Web Mercator Projection is specified as **EPSG:3857**.

2.3.3 Conversion Between Coordinates

The following equation roughly translates between the WGS to Mercator projection, which is close to the web Mercator Projection (one exception being results at the poles). Tools used in the analysis make a precise calculation - this equation is simplified, but emphasizes the importance of watching coordinate systems.

$$x = R(\lambda - \lambda_0) \tag{1}$$

$$y = R\ln(tan(\frac{\pi}{4} + \frac{\phi}{2})(\frac{1 - e\sin\phi}{1 + e\sin\phi})^{\frac{e}{2}} \tag{2}$$

$$k = \sec\phi\sqrt{1 - e^2\sin^2\phi} \tag{3}$$

- ϕ is latitude in degrees
- λ is longitude in degrees
- R is radius of Earth
- k is scale factor that changes by latitude

2.4 Analysis Methods

This analysis uses three primary methods to process and categorize the terrain profiles: The Continuous Wavelet Transform CWT) for edge filtering, the Fast Fourier Transform (FFT) for extracting frequency component independent of location, and Singular Value Decomposition (SVD) for correlations between the frequency components.

2.4.1 Continuous Wavelet Transform (CWT)

The Continuous Wavelet Transform (CWT) is a tool that can provide a representation of both the frequency spectrum and location components in a signal under analysis. It accomplishes this through "sweeping" a wavelet $\psi(t)$ across a signal while varying a scale value a and translation value b.

$$X_w(a,b) = \frac{1}{|a|^{1/2}}\int_{\text{inf}}^{\text{inf}} x(t)\psi(\frac{t-b}{a})dt \tag{4}$$

For this use case, we use the 2D Mexican hat wavelet kernel due to its applicability for filtering a signal, such as terrain profile, with sharp edges.

$$\psi(x,y) = \frac{1}{\pi\sigma^4}(1 - \frac{1}{2}(\frac{x^2+y^2}{\sigma^2}))e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5}$$

Analysis Note: Although the CWT has the excellent property of providing the location of a frequency, we only use it as a filtering step before using FFT to remove the location information. This is due to the fact we are attempting to analyze the terrain as a whole. If a canyon is located to the north or south of you, it should make no difference in how you would categorize the area - "a place with an awesome canyon."

2.4.2 Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is an algorithm to quickly perform a general Discrete Fourier Transform (DFT). The following definition is therefore a DFT specified to the standard FFT formula (used within software packages such as `Matlab` and `Numpy`).

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-\frac{i2\pi}{N}kn} \tag{6}$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] \cdot e^{\frac{i2\pi}{N}kn} \tag{7}$$

- $X[k]$ is an array with the signal's frequency domain data points
- $x[n]$ is an array with the signal's time domain data points
- n is an array with the time domain axis points
- k is an array with the ordinary frequency domain axis points
- N is the length of the time domain array, indicating number of time domain points, or signal size
- $n \in [0, N-1]$
- $k \in \frac{1}{N}[0, N-1]$, which is transformed around zero to represent frequency symmetry (for real signals), known as "fft shift":
  - $k \in \frac{1}{N}[-\frac{N}{2}, \frac{N}{2} - 1]$ (if N is even, with one extra data point in negative spectrum)
  - $k \in \frac{1}{N}[-\frac{N-1}{2}, \frac{N-1}{2}]$ (if N is odd, with equal data points between negative and positive spectrum)

In the context of geography the FFT will return the spacial frequency (m^{-1})

Analysis Note: For the use of the FFT to find the primary "signature" frequencies of a region, the orientation should not make a difference. We need a method to combine the results in the two dimensions. Although performing FFT in polar coordinates around the center of an image might be more appropriate (if that is even possible), in this case

we use average the absolute value of the FFT in both dimensions to form a composite signal.

2.4.3 Singular Value Decomposition (SVD)

We will use Singular Value Decomposition (SVD) to compare the various frequency samples.

The SVD performs this correlation by breaking down a matrix $\mathbf{M}$ into a standard set of components $\mathbf{U}$, Σ, and $\mathbf{V}$.

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V} \tag{8}$$

- $\mathbf{M}$ is a $n \times m$ matrix including all of the data to be transformed
- $\mathbf{V}$ is a $n \times n$ unitary matrix representing the *original* basis
- $\mathbf{U}$ is a $m \times m$ unitary matrix representing the *new* basis, moved from $\mathbf{V}$ by $\mathbf{M}$
- Σ is a $m \times n$ diagonal matrix representing the singular values of $\mathbf{M}$

  Note 1: We will be using the "compact SVD," which cuts Σ to size $r \times r$, $\mathbf{U}$ to size $m \times r$ and $\mathbf{U}$ to size $n \times r$ where $r \leq \min\{m, n\}$. This simplifies the size of the matrices without losing information.

  Note 2: Typically in the SVD equation $\mathbf{V}$ is noted as $\mathbf{V}^*$. However, in our case $\mathbf{V} = \mathbf{V}^*$ since $\mathbf{M}$ is real.

This process of using the SVD is analogous to re-framing the various terrain frequency samples as a series of scaling and rotational operations. Once these scale and rotational operations are found, we can more directly locate correlations between datasets.

3 Algorithm Implementation and Development

This analysis takes on the following steps:

1. Locate and process coordinates for thru-hike trails at appropriate resolution.
2. Select appropriate locations along the trails to analyze terrain.
3. Download and import terrain data from selected points.
4. Perform Continuous Wavelet Transform (CWT) to filter out appropriate frequencies.
5. Perform Fast Fourier Transform (FFT) and collapse axes to generate frequency signatures.
6. Review the frequency signatures with Singular Value Decomposition (SVD).

3.1 Locate and process coordinates for thru-hike trails

Locating thru-hike coordinates for the selected hikes is not very challenging since the trails are quite famous. However, most of the coordinate data is provided in a segmented format that is helpful for navigation, but posed a challenge when trying to generate a single line path.

- PCT shapefile
- CDT shapefile

- AT shapefile

Since the coordinates I located were not completely sorted from start to finish, it became essentially a shortest path problem solvable by Dijkstra's algorithm. Fortunately I was able to find a version from StackOverflow to modify to generate a clean and sorted version of the coordinates. It is possible a clean dataset is also available that would negate this step. It was also important to subsample the data for efficient computing at a resolution appropriate for the analysis.

The results from this step are demonstrated in Fig 1.

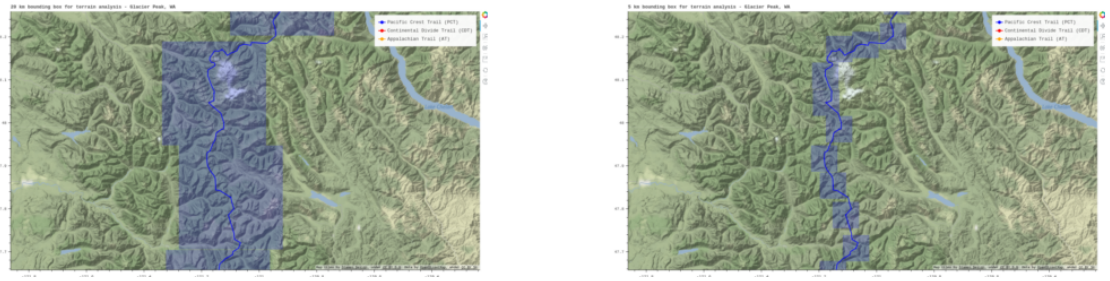
3.2 Select appropriate locations along the trails to analyze terrain

Selecting appropriate locations along the trail to sample requires a bit of subjective decisions to be made. The primarily criteria I considered were:

- Ensure there is no overlap between the terrain samples. An overlap would make any analysis difficult to parse due to strong correlations between the samples.
- Choose a distance between samples that allows enough samples for frequency analysis. Fortunately the trails are all quite long, so this was not a problem.
- Choose a distance appropriate to "experience" the terrain. This is the most subjective metric - if someone is on top of a mountain, they may be able experience hundreds of miles of the terrain at once. However, if someone is socked in with clouds, they may only experience a few meters of the terrain at once. Also, the terrain our of sight from a hiker will clearly influence the nearby terrain (e.g. a mountain will generate glaciers).

In order to understand the effect of terrain spacing two metrics were attempted (bounding distances of 20 km and 5 km). The focus of the analysis is on the 20 km spacing, which was initially more successful.

Fig 4. Bounding box examples for downloading data from trails (near Glacier Peak, WA)
20 km separation distance for bounding box (left); 5 km separation distance for bounding box (right)



3.3 Download and import terrain data from selected points

This step was made much easier by using the API provided by OpenTopology.com. If you have the coordinates for the extents of a bounding box to download, it is as simple as an API call to gather the DEM data. The main concern is to ensure that the calculations made for the bounding box are made in the correct coordinate system. A simplification was made to choose a consistent distance

as degree for the bounding boxes, allowing the same size file for each DEM file at the expense of some loss in bounding box distance accuracy.

> Note: One important decision made at this point was to remove the average elevation from each DEM file, essentially removing the idea of which specific elevation the sample was taken from. This was done to ensure the analysis is not picking up only the elevation of where a data point is taken - it would be trivial to say if you are very high in elevation that you are likely to be in Colorado. However, we do not want to fully normalize the data, since a nearby higher mountain peak is relevant to the local terrain.

3.4 Perform Continuous Wavelet Transform (CWT) to filter appropriate frequencies.

The Continuous Wavelet Transform (CWT) is used as a filtering method. It may be possible to build a targeted frequency based on the FFT spectrum, but it is has been much more efficient to perform the CWT and choose an appropriate level from the resulting coefficients as a filter.

We will review the CWT results of the Glacier Peak, WA for a mountainous region and a desert region in NM for a flat terrain sample to see how the filtering performs.

Fig 5. Terrain examples near Glacier Peak, WA
Before zeroing mean elevation (left); After zeroing mean elevation (middle); After zeroing mean elevation and fixing scale, used in analysis (right)

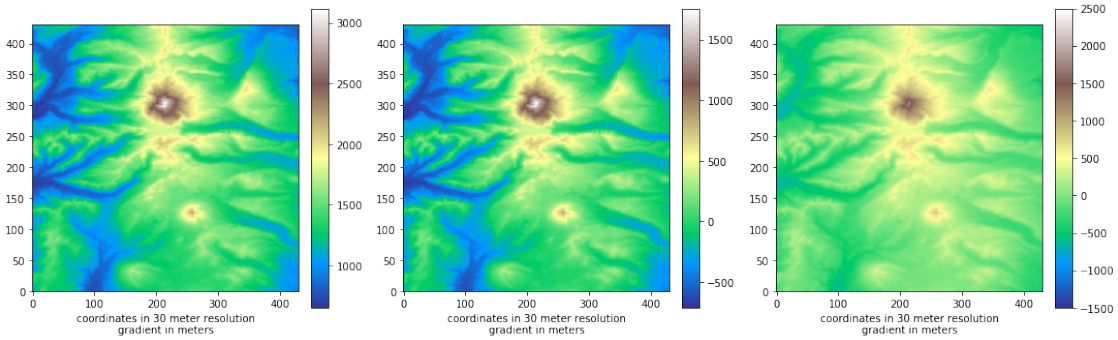
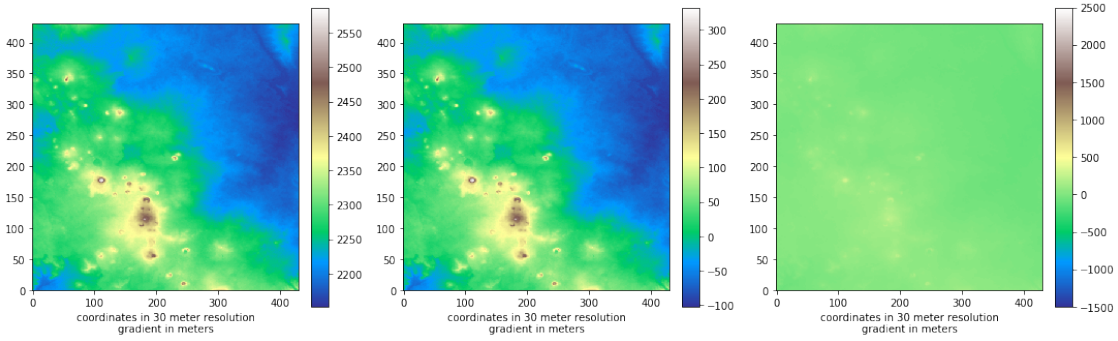


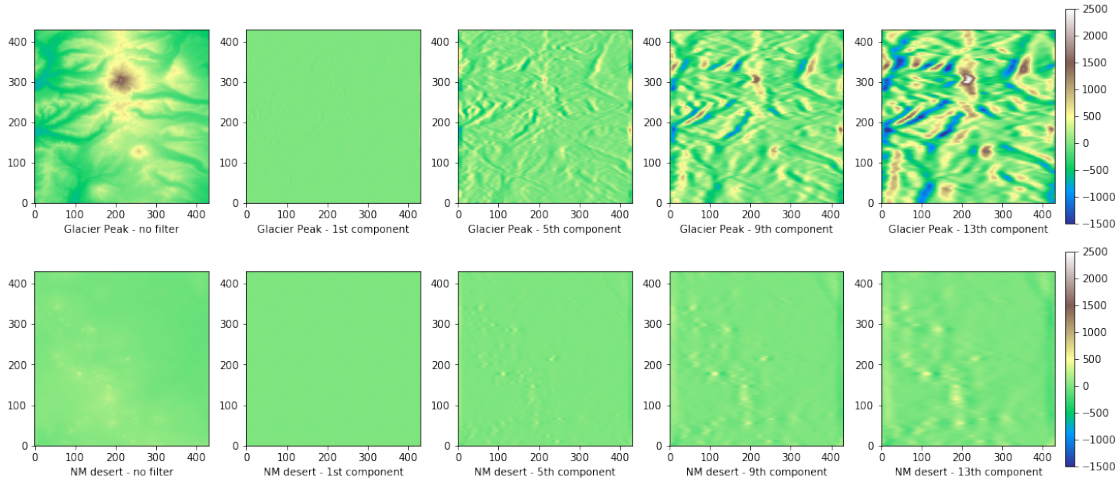
Fig 6. Terrain examples in New Mexico desert, WA
Before zeroing mean elevation (left); After zeroing mean elevation (middle); After zeroing mean elevation and fixing scale, used in analysis (right)



3.4.1 Selecting CWT component for filtering

Now that we have completed zeroing out the means, it looks like we have appropriate samples to review - a sample with a large variance and a sample with almost no variance. See Fig 6. for example CWT for filtering these samples.

Fig 7. Comparison of CWT filters for mountainous and flat regions
coordinates in 30 meter resolution; gradient in meters



We will select the 5th component filter as it appears to gather the most detail to differentiate and locate the shape of Glacier Peak while also limiting features on the desert. This parameter can be tuned to vary the geographic features being prioritized. Also note that the number of components for the CWT is much lower than preferred due to hardware limitations. It would be interesting to perform this analysis again with double or triple the number of CWT components to see if there is any effect.

We will review the frequency components of these two regions before and after the CWT filtering.

3.5 Perform Fast Fourier Transform (FFT) and collapse axes to generate frequency signatures

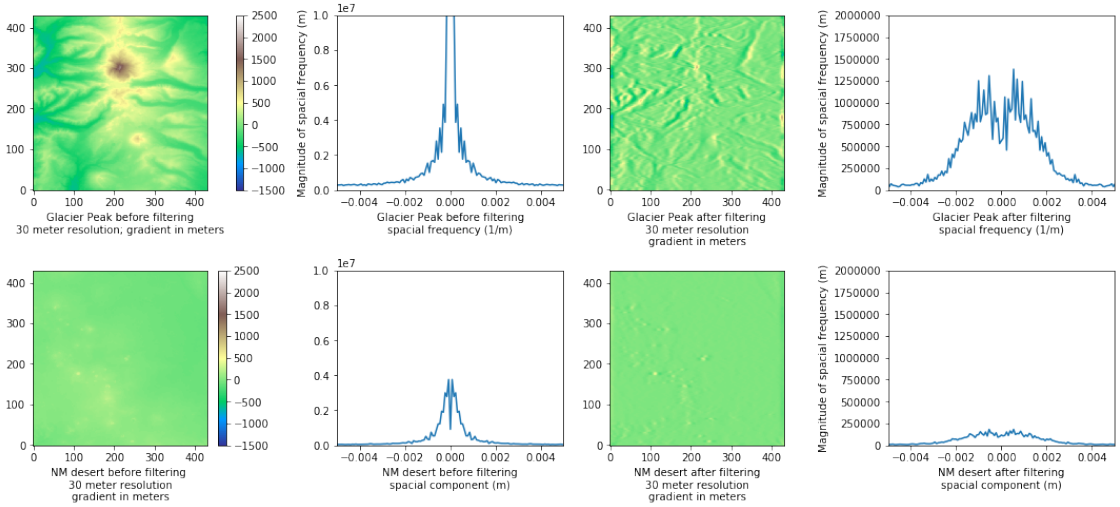
The Fast Fourier Transform (FFT) is used to remove any location information from the filtered terrain and focus only on the frequencies. It is assumed that a hiker on the trail will not be concerned with specifically which direction a terrain feature is located, since as the hiker moves the terrain will always be in a shifting relative location. While the CWT is great at picking out location, we want to remove that location information with the FFT.

We will review the FFT results of the filtered terrain from Glacier Peak, WA and a flat desert section in New Mexico.

After performing the FFT the large majority of the frequency spectrum approaches zero, and we must zoom in significantly on the frequency spectrum to see any useful results.

Note: The FFT results are combined together based on an average of the max frequencies in the x-direction and the max frequencies in the y-direction. This allows a simpler metric to review as a frequency key with similar information as full array of all FFT data.

Fig 8. Example frequncies retrieved from terrain data before and after Continous Wavelet Transform (CWT) filtering

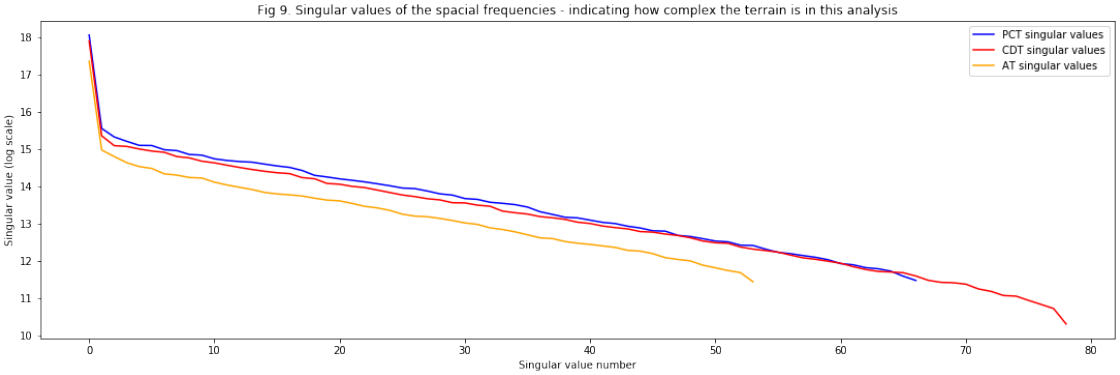


3.6 Review the frequency signatures with Singular Value Decomposition (SVD)

Finally, we pull together all of the terrain frequencies to locate which areas are the most unique in terms of the criteria selected in this analysis. Pulling out the "principal components" of the frequency data should be relatively straightforward.

Note that because we are pushing frequencies through the SVD, and furthermore frequencies that lack any spacial component, visualizing the U and V matrices would be difficult. We will therefore focus on the S matrix, which highlights how complex the terrain is for the thru-hike trails in this analysis.

Fig 9. Singular values of the spacial frequencies - indicating how complex the terrain is in this analysis



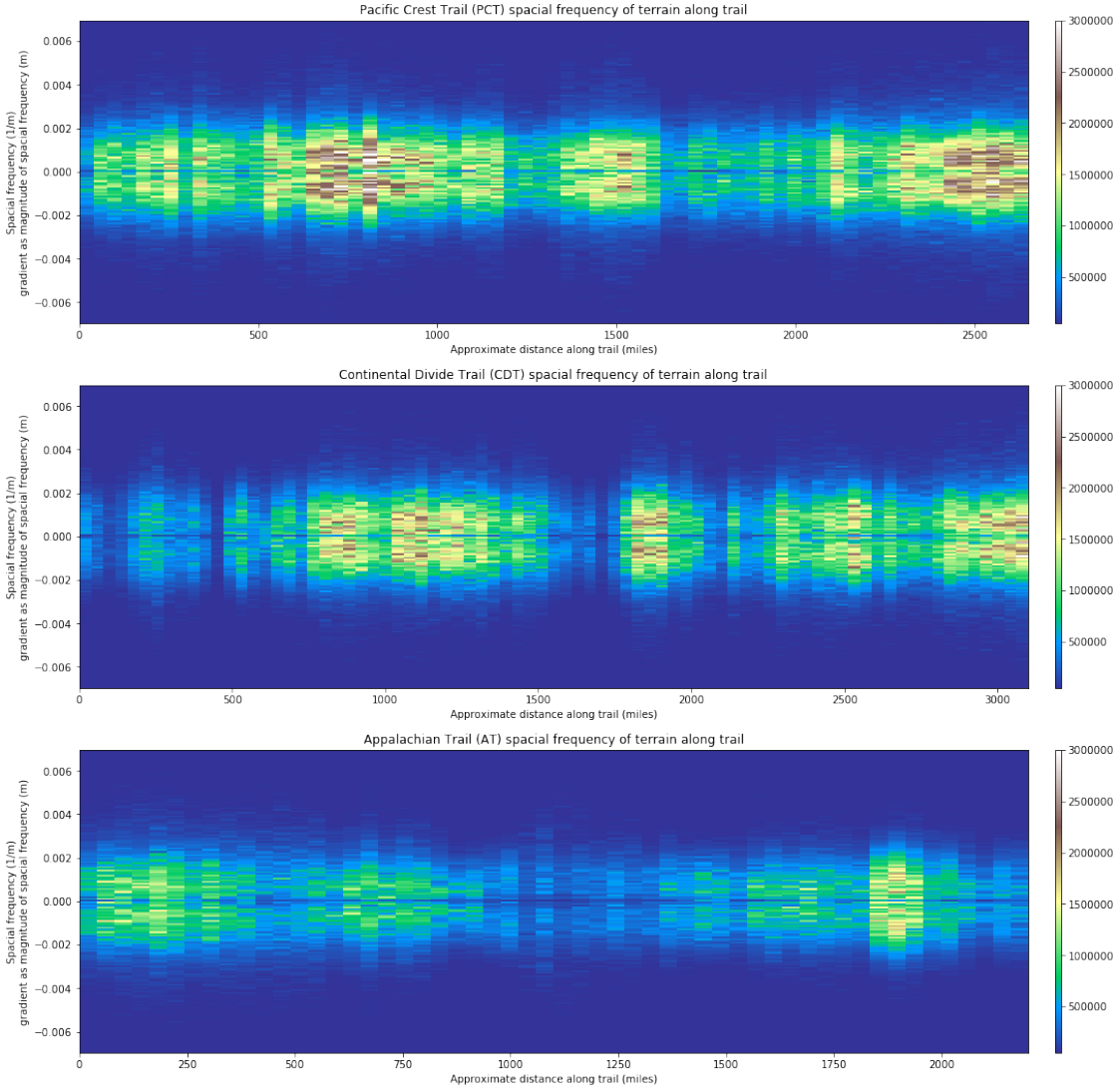
4 Computational Results

The breakdown of the singular values shows the relative complexity between the trails is similar, as one would expect. It also shows that the terrain complexity does vary depending on the thru-hike chosen. As per Figure 9, the PCT is rated as more complex than the CDT, and both of those trials are much more complex than the AT (note that this is a log scale plot).

The end result is interesting - I assumed that either the PCT or CDT would be more complex than the AT, but I wasn't sure which would be rated more complex between the two. I look forward to reviewing the maps more to understand which areas of complexity are worth exploring.

Finally, the summary in Figure 10 is a resource to review which areas of the trail contribute to the complexity seen in the singular value analysis. Although much of this intuition could be found by reviewing a map, it is interesting to see the figures for comparison.

Fig 10. Summary comparison of terrain complexity along thru-hikes



5 Summary and Conclusions

This has been a very interesting dive into geographic data. It is always said that 90% of the work in an analysis is getting the data formatted, and that is especially true when working with this kind of dataset. Although locating data and dealing with the coordinate transforms between geodesic and projected datasets was a large challenge, it was satisfying to start analyzing the data once it was collected.

One challenge with working with terrain data is how inherently correlated and connected each section of terrain is. It is an interesting challenge to try to find complexity in terrain based only on elevation data, when similar information could be found by checking other features like local weather trends, local vegetation cover, distance from a fault lines, and types of local ecosystems. In some sense working with only elevation data makes for a good analysis challenge but it is hard to say how useful it is when removed from other important data sources.

One thing this analysis has been able to show definitely is that transforms, such as CWT, FFT, and SVD are very useful in a variety of contexts. I was surprised how quickly an analysis of elevation terrain data began to resemble an analysis of images or video. These tools are able to process a very large amount of data quickly, and were able to reduce the complexity of a terrain analysis problem immensely.

And, if we all agree to rate a trail based on the terrain complexity as described within this very specific analysis, than we can say definitively we can all agree that the PCT is clearly the best thru-hike in the United States.

Appendix A (python packages)

package	function	description
numpy	`numpy.fft`	used for matrix operations including FFT
scipy	`linalg.svd`	used to perform SVD calculations, returns USV from matrix M
matplotlib	`matplotlib.pyplot`	used for plotting
pathlib	`pathlib.Path`	used to walk through folder structure to gather images
osgeo	`osgeo.gdal`	used for import and processing of DEM files
pyproj	`pyproj.transform`	used to transform between geometric projections
geopandas	`geopandas`	used to import and manipulate shapefiles
pywt	`pywt.cwt`	used for CWT edge filtering
sklearn	`sklearn.svm.SVC`	used as simple classifier
bokeh	`bokeh`	used for interactive visualizations

Appendix B (python code)

Github code https://github.com/micahg0/thru-hike-terrain-analysis/

See github for full code, the Jupyter notebook used to generate this report, and environment requirements to repeat this analysis.

```
import numpy as np
import logging
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.linalg import svd
from scipy.io import wavfile
from scipy import signal
from scipy.spatial.distance import pdist, squareform
import itertools
from pyproj import Proj, transform, Transformer
from pathlib import Path
import pywt
import subprocess
import os
import requests
from osgeo import gdal

import geopandas as gpd
import shapely
import geopy



logger = logging.getLogger()


class Trail:
    """ Class for working with data along trail, including shapefiles, Digital
Elevation Model (DEM) files, and downloading from OpenTopography.com API"""

    def __init__(self):
        self.shapefile_directory = None
        self.df = None
        self.coordinates = None
        self.file_path = None
        self.idx_start = None
        self.idx_stop = None
        self.coordinates_sampled = None
        self.bounding_box_set = None
        self.bounding_box_centers = None
        return
```

```
    def process_trail_from_shapefile(self, shapefile_directory, subsample_rate,
spacing):
        """Script to perform common actions, including shapefile, get
coordinates, subsample, and sort"""

self.load_shapefile_to_dataframe(shapefile_directory=shapefile_directory)
        self.get_coordinates_from_dataframe()
        self.subsample_coordinates(subsample_rate=subsample_rate)
        self.sort_coordinates()
        self.filter_coordinates_with_spacing(spacing=spacing)
        self.get_mapping_bounding_boxes()
        self.get_dem_bounding_boxes()


        return

    def load_shapefile_to_dataframe(self, shapefile_directory):
        """Load shapefile using `geopandas` and produce dataframe"""

        self.shapefile_directory = shapefile_directory
        self.df = gpd.read_file(self.shapefile_directory)
        logging.info('Loaded shapefile from {} with shape
{}.'.format(self.shapefile_directory, self.df.shape))
        self.df = self.df[-self.df.geometry.isnull()]
        logging.info('After removing any null geometry rows, df has shape
{}'.format(self.df.shape))

        return

    def get_coordinates_from_dataframe(self, map_crs="EPSG:3857"):
        """Extracts coordinates from dataframe. Defaults to using Web Mercator
Format (EPSG:3857)"""
        self.df = self.df.to_crs(map_crs)
        x = []
        y = []
        for geometry in self.df.geometry:

            if geometry.geom_type=='LineString':
                for coordinate in geometry.coords:
                    x.append(coordinate[0])
                    y.append(coordinate[1])
            elif geometry.geom_type=='MultiLineString':
                for linestring in geometry:
                    for coordinate in linestring.coords:
                        x.append(coordinate[0])
                        y.append(coordinate[1])
```

```
        self.coordinates = np.vstack((x, y))
        logging.info('Extracted coordinates with shape {} from dataframe
geomerty column (LineStrings and
MultiLineStrings)'.format(self.coordinates.shape))
        return

    def subsample_coordinates(self, subsample_rate=100):
        """ Samples a fraction of the full coordinate list. Defaults to sampling
1/100 samples"""

        self.coordinates = self.coordinates[:,::subsample_rate]
        logging.info('Coordinates reduced by factor of subsample rate {}, new
coordinates shape {}'.format(subsample_rate, self.coordinates.shape))
        return

    def sort_coordinates(self):
        """Sort coordinates based on most northern and most southern parts of
the trails (the beginning and ends)"""
        y_min = self.coordinates[1].min()
        y_max = self.coordinates[1].max()
        logging.info('Latitude (or y) min: {}; max: {}'.format(y_min, y_max))
        self.idx_start = self.coordinates[1].argmin()
        self.idx_stop = self.coordinates[1].argmax()

        logging.info('Starting coodinate (most southern) {} at index
{}'.format(self.coordinates[:,self.idx_start], self.idx_start))
        logging.info('Stopping coodinate (most northern) {} at index
{}'.format(self.coordinates[:,self.idx_stop], self.idx_stop))

        self.coordinates = self.find_gps_sorted(self.coordinates.T,
k0=self.idx_start, kend=self.idx_stop)
        # coordinates = coords_sorted.T.reshape(2,-1)
        logging.info('Coordinates sorted and trimmed into shape
{}'.format(self.coordinates.shape))
        self.coordinates_sampled = self.coordinates

        return


    def save_trail_coordinates(self, file_path):
        """Save trail coordinates to csv. Useful after processing and before
analysis"""

        self.file_path = file_path
        np.savetxt(file_path, self.coordinates)
        return
```

```
    def filter_coordinates_with_spacing(self, spacing):
        """Chooses coordinates separated by specific spacing. Allows no overlap
of test points"""

        self.spacing=spacing
        self.bounding_box_centers = []
        self.bounding_box_set = []
        logging.info('Filtering {} coordinates with spacing
{}'.format(self.coordinates.shape, spacing))
        for coord in self.coordinates.T:
            self.check_in_bounding_box_set(coord)

            if not self.in_bounding_box_set:
                bounding_box = self.get_bounding_box_coordinates(coord,
self.spacing)
                self.bounding_box_set.append(bounding_box)
                logging.debug('Added bounding box {} to bounding box set (now
has {} bounding boxes)'.format(bounding_box, len(self.bounding_box_set)))
                bounding_box_center = (coord[0], coord[1])
                self.bounding_box_centers.append(bounding_box_center)
                logging.debug('Added bounding box center {} to bounding box
centers (now has {} bounding boxe centers)'.format(bounding_box_center,
len(self.bounding_box_centers)))

        self.coordinates = np.array(self.bounding_box_centers).T
        logging.info('Coordinates shape filtered with spacing {}; new shape
{}'.format(self.spacing, self.coordinates.shape))
        return


    def check_in_bounding_box_set(self, coord):
        """Checks if the coordinate is in the current bounding box set. Returns
True or False"""

        logging.debug('Checking if {} is in bounding box set'.format(coord))
        self.in_bounding_box_set = False
        for bbox in self.bounding_box_set:
            if coord[0] < bbox[0][0] and coord[0] > bbox[1][0] and coord[1] <
bbox[0][1] and coord[1] > bbox[1][1]:
                self.in_bounding_box_set = True

        logging.debug('Coordinate {} is set: {}'.format(coord,
self.in_bounding_box_set))

        return self.in_bounding_box_set

    def get_mapping_bounding_boxes(self):
        """Retreives bounding boxes for use in plotting coordinates on web maps,
```

```
specifically in Bokeh"""

        north = self.coordinates[1] + self.spacing
        south = self.coordinates[1] - self.spacing

        east = self.coordinates[0] + self.spacing
        west = self.coordinates[0] - self.spacing

        self.mapping_bounding_boxes = np.vstack((west, east, south, north))
        logging.info('Calculated bounding boxes for plotting on web maps. First
sample {}; shape {}'.format(self.mapping_bounding_boxes[:,0],
self.mapping_bounding_boxes.shape))
        return

    def get_dem_bounding_boxes(self):
        """Retreives bounding boxes for use in downloading Digital Elevation
Model (DEM) files from OpenTopography.com"""

        inProj = Proj('epsg:3857')
        outProj = Proj('epsg:4326')
        self.spacing_degrees = self.spacing/111320
        coord_degrees = transform(inProj, outProj, self.coordinates[0],
self.coordinates[1], always_xy=True)

        wgs_north = coord_degrees[1] + self.spacing_degrees
        wgs_south = coord_degrees[1] - self.spacing_degrees

        wgs_east = coord_degrees[0] + self.spacing_degrees
        wgs_west = coord_degrees[0] - self.spacing_degrees

        self.dem_bounding_boxes = np.vstack((wgs_west, wgs_east, wgs_south,
wgs_north))
        logging.info('Calculated bounding boxes for download DEM files. First
sample {}; shape {}'.format(self.dem_bounding_boxes[:,0],
self.dem_bounding_boxes.shape))
        logging.debug('east-west:\n{}\north-south:\n{}'.format(wgs_east-
wgs_west, wgs_north-wgs_south))

        return

    def download_coordinates(self, relative_directory):
        """Downloads multiple Digital Elevation Model (DEM) files coordinate
list"""


        self.relative_directory = relative_directory
        logging.info('Starting to download {} DEM files to
{}'.format(self.dem_bounding_boxes.shape, self.relative_directory))
```

```
        for i in range(0, self.dem_bounding_boxes.shape[1]):
            bbox = self.dem_bounding_boxes[:,i]
            file_name = 'out-{:03d}.img'.format(i)
            file_path = os.path.join(self.relative_directory, file_name)
            self.download_dem_from_opentopography(bbox=bbox,
file_path=file_path)
            logging.info('Downloaded {} to {}'.format(i, file_path))
        return

    @staticmethod
    def download_dem_from_opentopography(bbox, file_path):
        """Downloads individual Digital Elevation Model (DEM) file from
OpenTopography.com API"""

        demtype='SRTMGL3'
        URL = 'https://portal.opentopography.org/otr/getdem'
        outputFormat = 'IMG'
        west = bbox[0]
        east = bbox[1]
        south = bbox[2]
        north = bbox[3]
        logging.info('Downloading bounding box {}'.format(bbox))
        logging.info('West: {}, Easth {}, South {}, North {}'.format(west, east,
south, north))
        PARAMS={'demtype': demtype,
                'west': west,
                'east': east,
                'south': south,
                'north': north}
        r = requests.get(url = URL, params = PARAMS)
        logging.info('Response header content type: {}; Code
{}'.format(r.headers['Content-Type'], r.status_code))
        with open(file_path, 'wb') as f:
            for block in r.iter_content(1024):
                f.write(block)
        logging.info('Downloaded DEM of {} to file {}'.format(bbox, file_path))
        return


    @staticmethod
    def get_bounding_box_coordinates(coord, spacing):
        """Returns bounding box spacing around coordinate"""

        distance = spacing*2
        north = coord[1] + distance
        south = coord[1] - distance

        east = coord[0] + distance
```

```
        west = coord[0] - distance

        bounding_box_coordinates = [(east, north), (west, south)]
        logging.debug('Bounding box coordinates returned:
{}'.format(bounding_box_coordinates))

        return bounding_box_coordinates



    @staticmethod
    def find_gps_sorted(xy_coord, k0=0, kend=-1):
        """Find iteratively a continuous path from the given points xy_coord,
        starting by the point indexes by k0;
        modified from https://stackoverflow.com/questions/31456683/plot-line-
from-gps-points"""


        N = len(xy_coord)
        distance_matrix = squareform(pdist(xy_coord, metric='euclidean'))
        mask = np.ones(N, dtype='bool')
        sorted_order = np.zeros(N, dtype=np.int)
        indices = np.arange(N)

        i = 0
        k = k0
        while True:
            if k==kend:
                break
            sorted_order[i] = k
            mask[k] = False

            dist_k = distance_matrix[k][mask]


            indices_k = indices[mask]

            if not len(indices_k):
                break

            # find next unused closest point
            k = indices_k[np.argmin(dist_k)]

            # you could also add some criterion here on the direction between
consecutive points etc.
            i += 1
        #drop last point
        coords_sorted = xy_coord[sorted_order]
```

```
        # trim the last setments to remove issues
        coords_sorted= coords_sorted[:-10]
        return coords_sorted.T

class Terrain:
    """ Class for working with data along trail, including shapefiles, Digital
Elevation Model (DEM) files, and downloading from OpenTopography.com API"""

    def __init__(self):
        self.dem_directory = None
        self.coordinates = None
        return

    def load_dem_directory(self, dem_directory, file_limit=False, dem_size =
431*431):
        """Loads Digital Elevation Model (DEM) files from directory into
array"""
        self.file_limit = file_limit
        self.dem_size = dem_size
        self.dem_directory = dem_directory
        self.num_files = len(os.listdir(dem_directory))
        logging.info('DEM directory has {} files'.format(self.num_files))
        if self.file_limit is False:
            self.X = np.zeros((self.num_files, self.dem_size))
            dem_files = Path(self.dem_directory).rglob('*.img')
            logging.info('Loading DEM files with size {} from {} with no file
limit'.format(self.dem_size, self.dem_directory))
        else:
            self.X = np.zeros((self.file_limit, self.dem_size))
            dem_files =
itertools.islice(Path(self.dem_directory).rglob('*.img'), self.file_limit)
        i=0
        for dem_path in dem_files:
            geo = gdal.Open(str(dem_path))
            arr = geo.ReadAsArray()
            logging.debug('Add array with shape {}'.format(arr.shape))
            arr = np.array(arr).flatten()
            self.X[i] = arr
            logging.debug('Processed {} image from from {}'.format(i, dem_path))
            i+=1

        logging.info('Load files from {} to array with shape
{}'.format(self.dem_directory, self.X.shape))

        return

    def subtract_mean_from_dem(self):
        """Zeros out the mean from the Digital Elevation Model (DEM) data to
```

```
remove impact of specific elevation"""

        self.mean_elevations = self.X.mean(axis=1)
        self.X = self.X - self.mean_elevations[:, None]

        return

    def get_fft_before_filter(self, sampling_period=30):
        """Return FFT based on Digital Elevation Model (DEM) data before
filtering"""

        self.n = int(np.sqrt(self.dem_size))
        # self.Xf = np.copy(self.X)
        # self.Xfs = np.copy(self.X)

        # logging.info('Copy X to Xf with shape {}'.format(self.Xf.shape))
        # for i in range(0, self.Xf.shape[0]):
        #     Xf_ = np.fft.fft2(self.X[i].reshape(self.n, self.n))
        #     self.Xf[i] = Xf_.flatten()
        #     self.Xfs[i] = np.fft.fftshift(Xf_).flatten()
        self.Xf = np.fft.fft(self.X, axis=1)
        self.Xfs = np.fft.fftshift(self.Xf)
        self.f = np.fft.fftfreq(n=self.n, d=30)
        self.fs = np.fft.fftshift(self.f)
        logging.info('Created Xfs with shape {}'.format(self.Xfs.shape))

        ax1_mean = self.Xfs.reshape(-1, 431,431).max(axis=1)
        ax2_mean = self.Xfs.reshape(-1, 431,431).max(axis=2)

        self.Xfs_comp = (ax1_mean + ax2_mean)/2
        logging.info('Created Xfs_comp with shape
{}'.format(self.Xfs_comp.shape))

        return

    def get_cwt(self):
        """Return Continuous Wavelet Transform (CWT) cooefficients on Digital
Elevation Model (DEM) using Mexican Hat Wavelet"""

        self.cA, self.cD = pywt.cwt(self.X, wavelet='mexh',
scales=np.arange(1,15))

        return


    def get_fft_after_filter(self, component=5):
        self.n = int(np.sqrt(self.dem_size))
```

```
        self.Cf = np.fft.fftn(self.cA[component], axes=[1])
        self.Cfs_comp = []
        for i in range(0, self.Cf.shape[0]):
            arr = self.Cf[i]
            arr_reshape = np.fft.fftshift(np.real(arr.reshape(self.n, self.n)))
            arr_mean1 = arr_reshape.max(axis=0)
            arr_mean2 = arr_reshape.max(axis=1)
            arr_mean = (arr_mean1 +  arr_mean2)/2
            self.Cfs_comp.append(arr_mean)
            # ax2.reshape(-1, 431,431).max(axis=1)
            # ax2_mean = self.Cfs.reshape(-1, 431,431).max(axis=2)
        logging.info('Created Cfs_comp with length
{}'.format(len(self.Cfs_comp)))
        self.Cfs_comp = np.array(self.Cfs_comp)
        return


if __name__ == "__main__":

    import matplotlib.pyplot as plt
    import glob
    import os
    import sys
    logger = logging.getLogger()
    # logger.setLevel(logging.DEBUG)
    logger.setLevel(logging.INFO)

    logging.info(os.getcwd())
    os.chdir('./AMATH-582/project/')
    logging.info(os.getcwd())

    os.system('mkdir -p ./test/trail-shapefiles')
    # Unzip and prepare shapefile directories
    # !mkdir -p ./test/trail-shapefiles
    # !unzip -o -q ./data/imported/stelprdb5332131.zip -d ./test/trail-
shapefiles/pct/
    # !unzip -o -q ./data/imported/CDTbyState20161128.zip -d ./test/trail-
shapefiles/cdt/
    # !unzip -o -q ./data/imported/AT_Centerline_12-23-2014.zip -d ./test/trail-
shapefiles/at/

    shapefile_directory = './test/trail-shapefiles/pct'


    # shapefile_directory = './local/trails/CDTbyState20161128'
    # shapefile_directory = './local/trails/AT_Centerline_12-23-2014'
```

```
    # trail = Trail()
    # trail.load_shapefile_to_dataframe(shapefile_directory=shapefile_directory)
    # # trail.get_coordinates_from_dataframe(map_crs="EPSG:4326")
    # trail.get_coordinates_from_dataframe()
    # trail.subsample_coordinates()
    # trail.sort_coordinates()
    # trail.save_trail_coordinates('./test-coords.csv')
    # plt.plot(trail.coordinates[0], trail.coordinates[1])
    # plt.savefig('./test-img.png')

    # trail.filter_coordinates_with_spacing(spacing=1000)
    # # trail.filter_coordinates_with_spacing(spacing=20000)

    # trail.get_mapping_bounding_boxes()
    # trail.get_dem_bounding_boxes()


    # test_bbox = trail.dem_bounding_boxes[:,1]
    # trail.download_dem_from_opentopography(bbox=test_bbox,
file_path='./test.img')
    # trail.dem_bounding_boxes = trail.dem_bounding_boxes[:,:3]

    # trail.download_coordinates(relative_directory='./local/test-dem/')
    # trail2 = Trail()
    # trail2.process_trail_from_shapefile('./local/trails/CDTbyState20161128',
subsample_rate=1000, spacing=20e3)
    # trail2.dem_bounding_boxes = trail2.dem_bounding_boxes[:,:10]
    # Path('./test/dem/cdt-20000/').mkdir(parents=True, exist_ok=True)

    # trail2.download_coordinates('./test/dem/cdt-20000/')

    ter = Terrain()

    ter.load_dem_directory('./test/dem/cdt-20000/', file_limit=False)
    ter.load_dem_directory('./test/dem/cdt-20000/', file_limit = 20)
    ter.subtract_mean_from_dem()
    ter.get_fft_before_filter()
    ter.get_cwt()
```