

This notebook shows some examples of performing spectral filtering

```
In [1]: import sys
sys.path.append('../demAnalysisComponents') #Add the path to the demAnalysis scripts

import dem as dpy
import fftGrid as fg
from matplotlib import pyplot as plt
import numpy as np

from importlib import reload
```

```
In [2]: reload(dpy)
reload(fg)

%matplotlib notebook
```

```
In [3]: #Define a path to some data to test out functions
testDataPath = '../testData/rasterData/OD_10m_MicroTest.tif'

#Create an instance of a dem grid object
testDEM = dpy.demGrid([], rasterPath = testDataPath)
```

```
In [4]: '''To conduct spectral filtering, create an fftGrid instance.'''

#Create a specDEM from an existing DEM
specDEM = fg.fftGrid(testDEM)

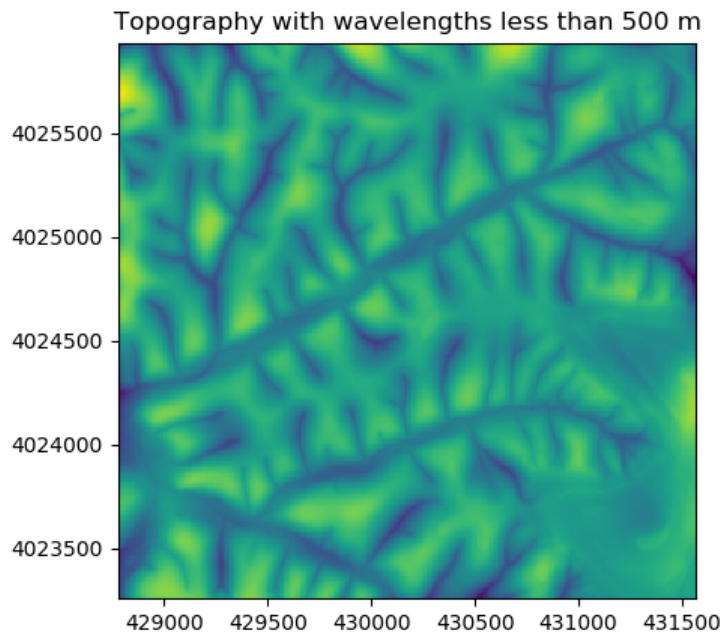
#Add a filtering operation, Lets Look at the short wavelength features
maximumWavelength = 500
sigma_w = 30 #This parameter will smooth the limit of wavelengths with a gaussian distribution, its optional
specDEM.addFilter_highPass(maximumWavelength,sigma_wavelength = sigma_w)

#Apply the filters
specDEM.applyFilters()

#Reverse the transform
filteredDEM = specDEM.inverseTransform().removePlaneFromGrid()

#Plot the filtered product
axs = filteredDEM.plotGrid()
axs.set_title('Topography with wavelengths less than {} m'.format(maximumWavelength))

../demAnalysisComponents/fftGrid.py:231: RuntimeWarning: divide by zero encountered in true_divide
  self.L = 1.0 / np.sqrt(FX*FX + FY*FY)
```



```
Out[4]: Text(0.5, 1.0, 'Topography with wavelengths less than 500 m')
```

```

In [7]: '''Can conduct basic standard filtering operations'''
f,axs = plt.subplots(4,1,figsize = (5,12),dpi = 120)

#First set some limits for colormaps
minZ = np.min(testDEM.grid)
maxZ = np.max(testDEM.grid)

#Define two bounding wavelengths to look at components between that
lowWavelength = 200
highWavelength = 600

#Top plot - the original grid
testDEM.plotGrid(axs = axs[0],vmin = minZ,vmax = maxZ)
axs[0].set_title('Original topography')

#Second plot - the long wavelengths of the topography
lp = fg.fftGrid(testDEM)
lp.addFilter_lowPass(highWavelength,sigma_wavelength = 50)
lp.applyFilters()
lpDEM = lp.inverseTransform()
lpDEM.plotGrid(axs = axs[1],vmin = minZ, vmax = maxZ)
axs[1].set_title('Topography with wavelengths greater than {} m'.format(highWavelength))

#Third plot - the middle wavelengths of the topography
bp = fg.fftGrid(testDEM)
bp.addFilter_bandPass(lowWavelength,highWavelength,sigma_wavelength = (lowWavelength-highWavelength)/2)
bp.applyFilters()
bpDEM = bp.inverseTransform()
bpDEM.plotGrid(axs = axs[2],vmin = minZ, vmax = maxZ)
axs[2].set_title('Topography with wavelengths in the range {}-{} m'.format(lowWavelength,highWavelength))

#Fourth plot - the short wavelengths of the topography
hp = fg.fftGrid(testDEM)
hp.addFilter_highPass(lowWavelength,sigma_wavelength = 50)
hp.applyFilters()
hpDEM = hp.inverseTransform()
hpDEM.plotGrid(axs = axs[3],vmin = minZ, vmax = maxZ)
axs[3].set_title('Topography with wavelengths less than {} m'.format(lowWavelength))

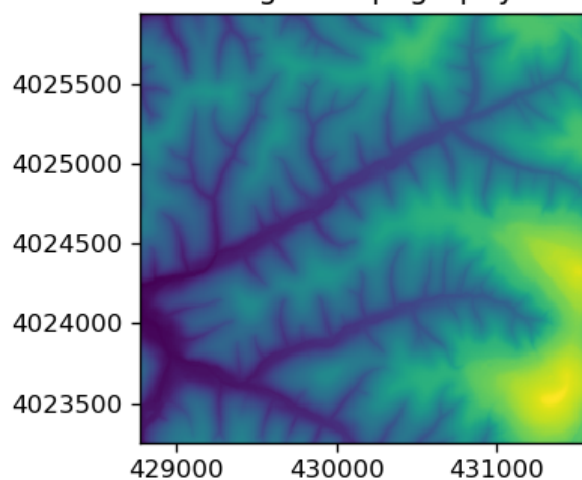
plt.tight_layout()

```

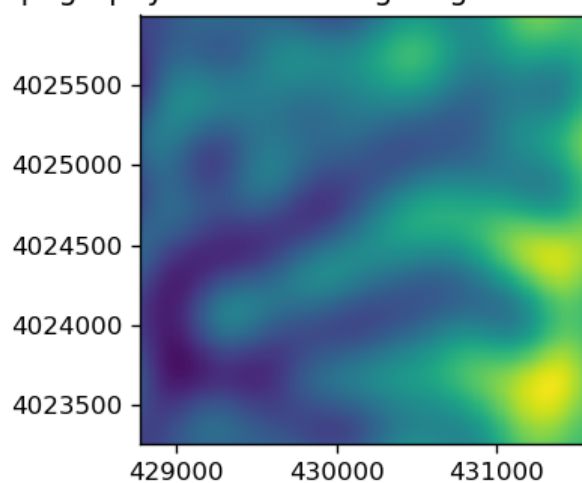




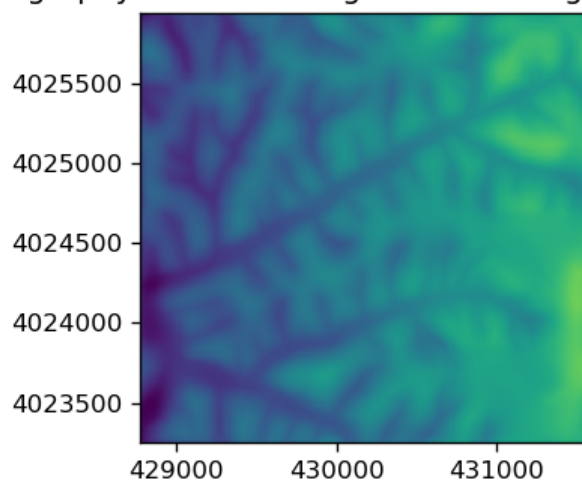
Original topography



Topography with wavelengths greater than 600 m

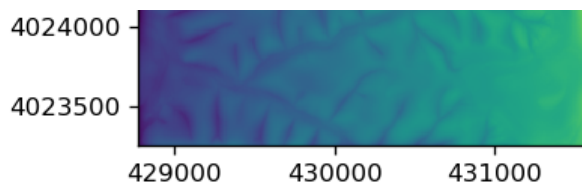


Topography with wavelengths in the range 200-600 m



Topography with wavelengths less than 200 m





In [9]: *'''Now Lets try looking at data radially'''*

```
#Apply the hann window to the original data to minimize ringing
#(NOTE: this will make it impossible to recover non-central parts of the grid)
newSpecDEM = fg.fftGrid(testDEM, applyHannWindow=True)

#Large grids will have a lot of individual points, we can plot a fraction of those observations with the
#'plotFractionOfObservations' parameter (set to 1 for all data). This will choose a random subset of data to s
how.
#However, because most data points are at short wavelengths, we will try to distribute this random subset more
evenly
#across different lengths to show the general structure of the power spectra
axs = newSpecDEM.plotPowerWavelength(plotFractionOfObservations=0.1,alpha = 0.05, color = 'k',zorder = -1)

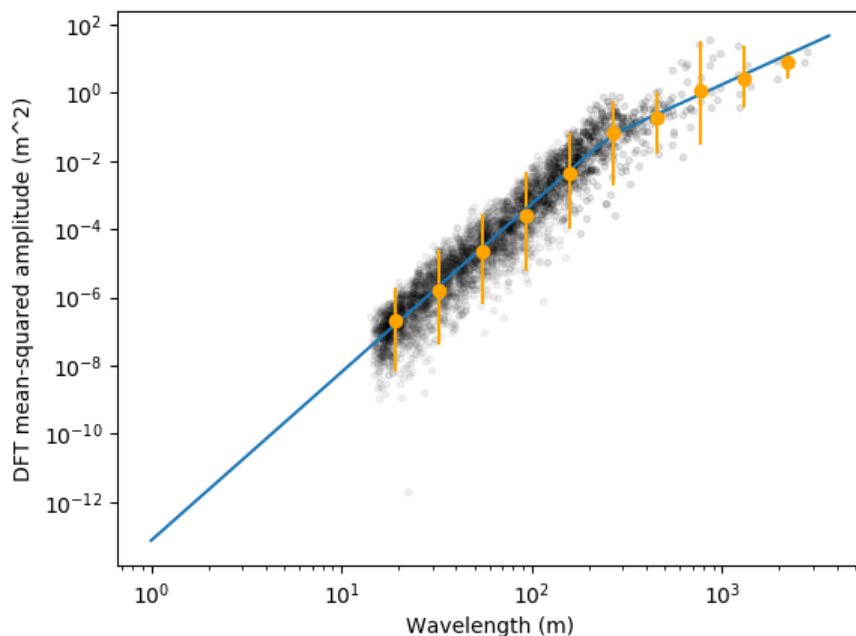
#Can also plot the median (and 95% percentile) of binned observations
newSpecDEM.plotMedianPowerWavelength(axs = axs,nBinsForMedian= 10,color = 'orange')

xRange = axs.get_xlim()

# axs.set_ylim(1e-9,1e2)

#Often data display a scaling break, we can fit power-laws to the data to try and sort out where that occurs
piecewisePLfit = newSpecDEM.calcPiecewiseRegressionCoefficients()

LsToPlot = np.logspace(0,np.log10(xRange[1]),20)
axs.plot(LsToPlot,newSpecDEM.calcPiecewiseRegressionPredictions(LsToPlot,piecewisePLfit))
```



Out[9]: [*<matplotlib.lines.Line2D at 0x13245c1d90>*]

```

In [10]: '''
We are really after anomalies in spectral power (e.g., where are those two 'bumps' in the above plots?).

To identify these, we are going to adopt the approach of Perron et al., 2008 JGR, where we are going to
create an 'expected' fractal distribution to normalize by that is random (e.g., without any particular
structure to it). We will do this with the 'diamond square' algorithm, an approach to creating random, fractal
topography with minimal parameters. The main parameter of interest here is the 'roughness', a value referred to
as 'H'
that can vary between 0 and 1, and that we want to tune to be close tho the mean expectation of our power-spec
tra.
'''

#Identify the best fitting 'roughness' value for the diamond square algorithm
reload(dpy)
import randomGrids as rg
reload(rg)

nMedianBins = 20

#Create an instance of a dem grid object
testDEM = dpy.demGrid([], rasterPath = testDataPath)

#Apply the hann window to the original data to minimize ringing
#(NOTE: this will make it impossible to recover non-central parts of the grid)
newSpecDEM = fg.fftGrid(testDEM, applyHannWindow=True)

#Fit roughness to the DEM by creating some test random DEMs
H_fit = newSpecDEM.calcBestFittingDiamondSquareRoughness(nBinsForMedian = nMedianBins, nRoughnessValuesToTest =
15)

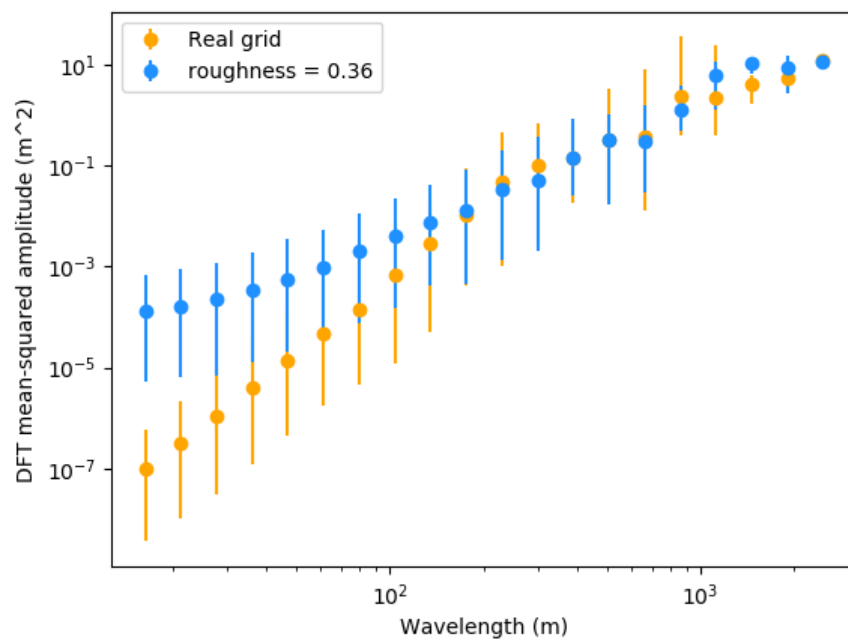
#Plot roughness comparison
dsq = rg.proceduralGrid_diamondSquare(testDEM, roughness = H_fit, randomSeed= 10, matchMeanElevationAndRelief=False)
randInst = dsq.getPermutation()
fftRI = fg.fftGrid(randInst, applyHannWindow = True)

#Plot the binned wavelength power values for the random and real grids
f, axes = plt.subplots(1, 1)
newSpecDEM.plotMedianPowerWavelength(axes = axes, nBinsForMedian= nMedianBins, color = 'orange', label = 'Real grid')
fftRI.plotMedianPowerWavelength(axes = axes, nBinsForMedian = nMedianBins, color = 'dodgerblue', label = 'roughness = {:.2f}'.format(H_fit))
axes.legend()

```



```
../demAnalysisComponents/fftGrid.py:145: RuntimeWarning: divide by zero encountered in true_divide  
demGridInstance = (np.real(np.fft.ifft2(self.grid))/self._windowMultiplier) + self._detrendGrid  
../demAnalysisComponents/fftGrid.py:145: RuntimeWarning: invalid value encountered in true_divide  
demGridInstance = (np.real(np.fft.ifft2(self.grid))/self._windowMultiplier) + self._detrendGrid
```



Out[10]: <matplotlib.legend.Legend at 0x13259f5650>

```

In [33]: '''Check how roughness value impacts spectral power plots'''

from matplotlib import cm
reload(dpy)
reload(rg)
reload(fg)

#Create a list of roughness values that we would like to look at
roughnessToTest = np.linspace(0,1,5)

#Get a colormap to plot different power-spectra as
cmap = cm.get_cmap('viridis')
colors = [cmap(i) for i in np.linspace(0,1,len(roughnessToTest))]

#How many log-bins do we want to look at power spectra in?
nMedianBins = 10

#First, transform a grid to create some 'real' data for reference
newSpecDEM = fg.fftGrid(testDEM, applyHannWindow=True)

#Get the binned mean wavelengths and powers for that data
L_obs, P_obs = newSpecDEM.calcWavelengthBinnedMedianPower(nMedianBins)[-1]

#Create an axis to plot the results on
f,axs = plt.subplots(1,1)

#Loop through the roughness values we wanted to test
for i,H_i in enumerate(roughnessToTest):

    #Create an instance of the random grid generation routine with this roughness
    dsq = rg.proceduralGrid_diamondSquare(testDEM,roughness = H_i,matchMeanElevationAndRelief=False)

    #Get a single random grid based on the specified parameters
    randInst = dsq.getPermutation()

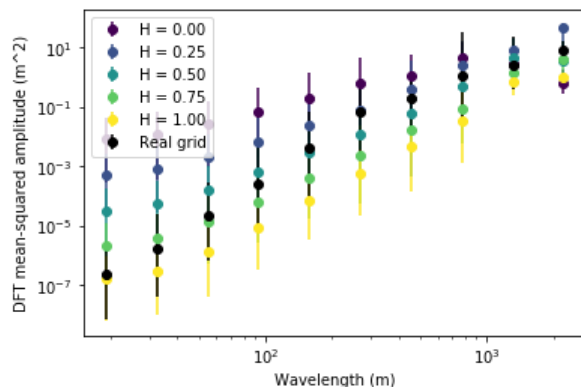
    #Preform the spectral transformation on that grid
    fftRI = fg.fftGrid(randInst,applyHannWindow = True)

    #Plot the spectral power as a function of wavelength for this random grid
    fftRI.plotMedianPowerWavelength(axs = axs, nBinsForMedian = nMedianBins, color = colors[i],
                                    label = 'H = {:.2f}'.format(H_i))

newSpecDEM.plotMedianPowerWavelength(axs = axs,nBinsForMedian= nMedianBins,color = 'k', label = 'Real grid')
axs.legend(loc = 'upper left')

```

Out[33]: <matplotlib.legend.Legend at 0x1320333810>



```

In [11]: #What we are really curious about is the directions and length-scales of anomalies in the power spectra
# we can use random grids to normalize spectral power plots, to better highlight major component waveforms
reload(fg)

#Create an instance of a dem grid object
testDEM = dpy.demGrid([], rasterPath=testDataPath)

#Get a fft dem
newSpecDEM = fg.fftGrid(testDEM, applyHannWindow=True)

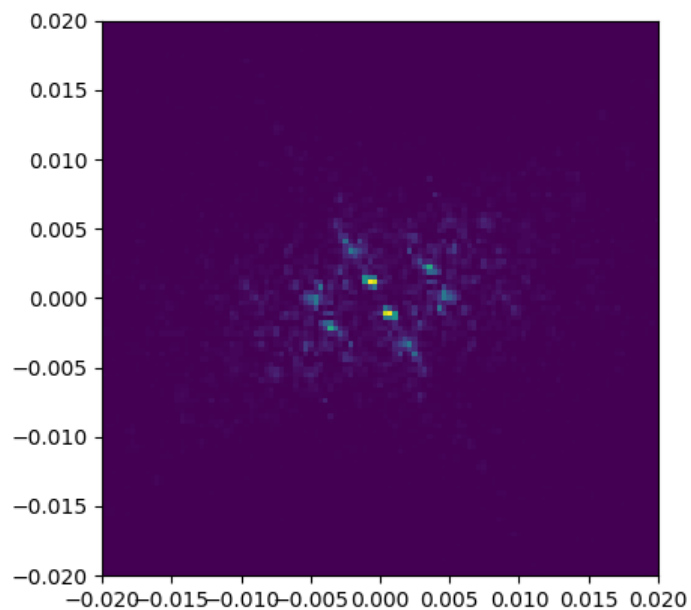
#Calculate a number of random permutations of topography using the diamond-square algorithm
nPermutations = 10#In practice this number should be large, thousands if your able

#Compare these values to the average of 50 of the random grids
#this will first find the best fitting roughness unless its specified
HtoUse = H_fit #This value obtained a couple cells up
specPowerPermutations = newSpecDEM.calcDiamondSquareComparisonAnomalyGrid(nPermutations = nPermutations,
                                                                           roughness=HtoUse)

#Plot the anomalies
axs, countGrid, normGrid = newSpecDEM.plotSpectralAnomalies(logTransform = False)

#Zoom in on the center of the image
wavelengthRangeToCenter = 50
axs.set_xlim(-1/wavelengthRangeToCenter, 1/wavelengthRangeToCenter)
axs.set_ylim(-1/wavelengthRangeToCenter, 1/wavelengthRangeToCenter)

```



Out[11]: (-0.02, 0.02)

```

In [13]: #We can also project these anomalies into a length, angle coordinate space to make things a bit easier to relate
#to observed patterns in Landscapes
reload(dpy)
reload(rg)
reload(fg)

#Set a roughness value (rather than searching for the best fit), this is about what was fit for the example grid
#in above cells
roughnessForComparison = 0.5

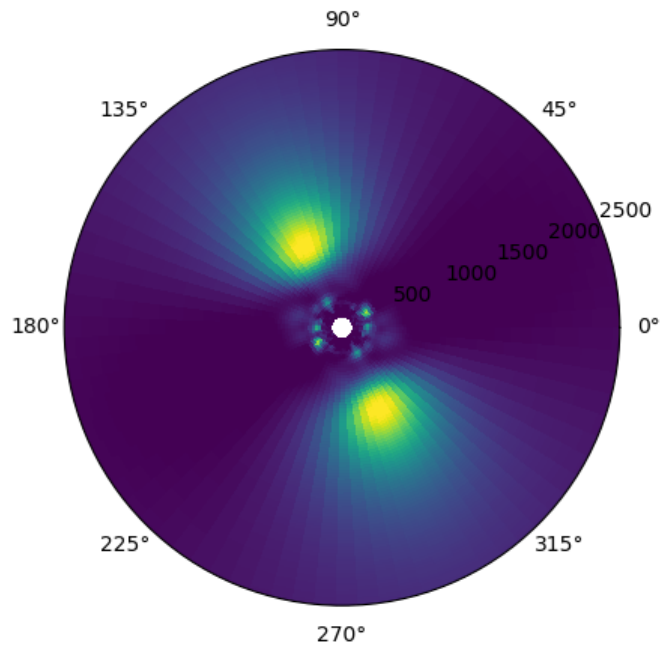
#Reload an instance of a dem grid object
# testDEM = dpy.demGrid(filePath = testDataPath)

#Get a fft dem
# newSpecDEM = fg.fftGrid(testDEM, applyHannWindow=True)

#Calculate a number of random permutations of topography using the diamond-square algorithm
nPermutations = 10#In practice this number should be large, thousands if your able - but this is slow
specPowerPermutations = newSpecDEM.calcDiamondSquareComparisonAnomalyGrid(roughness = roughnessForComparison,
                                                                           nPermutations = nPermutations)

axs = newSpecDEM.plotRadialSpectralAnomalies(Ls = np.linspace(100, 2500, 80), nThetaBins = 80, vmin = 3, vmax = 50)

```



```
In [18]: newSpecDEM._randomPowerGrids
```

```

Out[18]: array([[ 2.01522484e+02,  1.08170268e+02,  1.90022066e+00, ...,
                  2.20107984e+02,  5.95056592e+01,  3.94847116e+02],
                 [ 2.09527564e+02,  7.99973108e+01,  2.56768451e+01, ...,
                  2.49228247e+02,  3.42198351e+01,  1.88246236e+02],
                 [ 9.61494905e+01,  2.84267121e+01,  3.33801505e+01, ...,
                  1.64640676e+02,  2.85731825e+01,  2.11802575e+01],
                 ...,
                 [ 2.24249458e+01,  4.49445041e+00,  3.33014820e+01, ...,
                  2.95525751e+01,  1.98166662e+01,  1.31890948e+01],
                 [ 9.61494905e+01,  2.84267121e+01,  3.33801505e+01, ...,
                  1.64640676e+02,  2.85731825e+01,  2.11802575e+01],
                 [ 2.09527564e+02,  7.99973108e+01,  2.56768451e+01, ...,
                  2.49228247e+02,  3.42198351e+01,  1.88246236e+02]],

                [[ 6.23256940e+01,  1.01258259e+02,  4.22860786e+02, ...,
                  4.45405843e+02,  1.76097575e+02,  1.52962912e+03],
                 [ 1.07182905e+02,  1.00704861e+02,  3.65309141e+02, ...,
                  2.61400092e+02,  3.88311855e+01,  5.33238369e+02],
                 [ 1.00476400e+01,  1.59294030e+01,  4.57873529e+01, ...,
                  1.32265894e+02,  2.36722554e+00,  3.25109747e+01],
                 ...,
                 [ 2.13969522e+01,  4.34656624e+01,  3.14855317e+01, ...,
                  1.21030346e+01,  6.77543891e+00,  2.71278453e+01],
                 [ 5.59208896e+01,  1.20318450e+02,  9.66736258e+00, ...,
                  6.14862041e+01,  6.43314100e+01,  1.55249369e+02],
                 [ 3.15275081e+01,  1.30012285e+02,  1.06776377e+02, ...,
                  1.17528814e+02,  1.56622515e+02,  9.18980924e+02]],

                [[ 5.77243816e+01,  5.05085782e+01,  1.66959709e+02, ...,
                  1.05155004e+02,  2.10390419e+01,  3.99596821e+02],
                 [ 4.82062684e+01,  9.43536166e+01,  1.82604144e+02, ...,
                  2.97433618e+01,  3.29498225e+00,  1.34630078e+02],
                 [ 1.87060007e+01,  5.01964926e+01,  2.34870425e+01, ...,
                  1.88790682e+01,  1.67135878e+01,  8.00741255e+00],
                 ...,
                 [ 1.32879701e+01,  3.60696494e+01,  2.14281541e+00, ...,
                  4.26244833e+00,  3.81236189e+01,  2.26715018e+01],
                 [ 6.09639615e+00,  7.28420937e+01,  5.14800321e+00, ...,
                  1.32740744e+01,  7.67659189e+01,  2.46936538e+01],
                 [ 1.64350443e+01,  4.59926988e+01,  1.46936871e+01, ...,
                  2.47043604e+01,  4.58420919e+01,  1.90800163e+02]],

                ...,

                [[ 1.78026565e+01,  8.33488104e+00,  2.94737557e+01, ...,
                  4.56476809e-01,  4.22254291e-01,  5.60280388e+00],
                 [ 1.96409263e+01,  1.52870729e-01,  1.20732731e+01, ...,
                  2.20835948e+00,  1.47302146e+01,  4.36228172e+00],
                 [ 8.77915912e+00,  9.44468658e+00,  2.46248673e+00, ...,
                  1.40747867e+00,  2.23016309e+01,  2.32458351e+01],
                 ...,
                 [ 7.28723595e-01,  8.45014480e+00,  6.11647412e+00, ...,
                  2.18291716e+00,  6.11669727e+00,  1.87086619e+01],
                 [ 4.16951139e+00,  1.75999360e+01,  1.51436434e+01, ...,
                  1.19388408e+00,  2.86141531e+01,  9.64485742e+00],
                 [ 5.83027512e+00,  7.94542097e+00,  3.17826941e+01, ...,
                  5.03858721e+00,  4.91946544e+00,  9.53406776e+00]],

                [[ 5.77243816e+01,  5.05085782e+01,  1.66959709e+02, ...,
                  1.05155004e+02,  2.10390419e+01,  3.99596821e+02],
                 [ 1.64350443e+01,  4.59926988e+01,  1.46936871e+01, ...,
                  2.47043604e+01,  4.58420919e+01,  1.90800163e+02],
                 [ 6.09639615e+00,  7.28420937e+01,  5.14800321e+00, ...,
                  1.32740744e+01,  7.67659189e+01,  2.46936538e+01],
                 ...,
                 [ 1.11921401e+01,  1.17172100e+01,  8.97776100e+00, ...,
                  6.05355094e+00,  2.93153245e+00,  3.24688893e+01],
                 [ 1.87060007e+01,  5.01964926e+01,  2.34870425e+01, ...,
                  1.88790682e+01,  1.67135878e+01,  8.00741255e+00],
                 [ 4.82062684e+01,  9.43536166e+01,  1.82604144e+02, ...,
                  2.97433618e+01,  3.29498225e+00,  1.34630078e+02]],

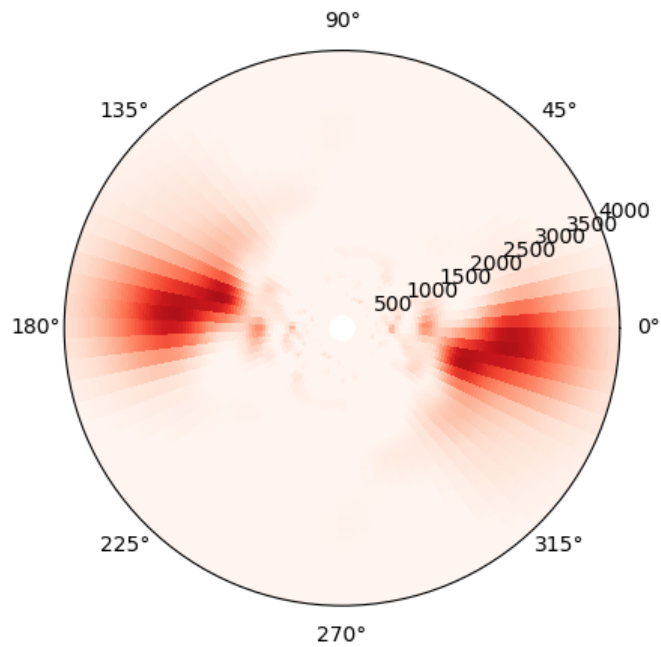
                [[ 6.23256940e+01,  1.01258259e+02,  4.22860786e+02, ...,
                  4.45405843e+02,  1.76097575e+02,  1.52962912e+03],
                 [ 3.15275081e+01,  1.30012285e+02,  1.06776377e+02, ...,
                  1.17528814e+02,  1.56622515e+02,  9.18980924e+02]],

```

```
[5.59208896e+01, 1.20318450e+02, 9.66736258e+00, ...,
 6.14862041e+01, 6.43314100e+01, 1.55249369e+02],
...,
[3.57506635e+00, 6.46175822e-01, 5.94782989e+00, ...,
 3.30696462e+01, 2.40726703e+01, 2.08894926e+01],
[1.00476400e+01, 1.59294030e+01, 4.57873529e+01, ...,
 1.32265894e+02, 2.36722554e+00, 3.25109747e+01],
[1.07182905e+02, 1.00704861e+02, 3.65309141e+02, ...,
 2.61400092e+02, 3.88311855e+01, 5.33238369e+02]]])
```

```
In [20]: #Below needs to be updated for this function
f, axs = plt.subplots(1, 1, subplot_kw=dict(projection='polar'))

#Plot the plot
axs.pcolor(newSpecDEM.Thetagrid, newSpecDEM.Lgrid, newSpecDEM._length_theta_anomalyGrid, vmin = 5, vmax = 75, c
map = 'Reds')
plt.savefig('RadialSpectralAnomaly.eps')
```



```
In [ ]:
```