Coding Project 1: Detecting objects through frequency signatures

Manjaree Binjolkar

Abstract

A Kraken's trajectory was estimated using noisy acoustic data with an unknown characteristic frequency. The spatial frequency and location of the submarine in 3D space were determined using Fourier analysis and filtering in Python. There was a significant improvement when the results were compared before and after filtering.

1 Introduction

One of the significant benefits of radar technology is its ability to detect objects through their frequency signatures. Different materials and objects have distinct frequency signatures, allowing radar systems to identify and locate them.

In this paper we will use these frequency signatures to track the movement of a Kraken that lurks beneath the ice at the Climate Pledge Arena. The vibrations created by the Kraken's movement is recorded using a spatio-temporal seismometer at 1 minute and 15 second intervals, and stored in Kraken.mat.

2 Theoretical Background

Data and noise frequently coexist, making it almost impossible to extract a noise-free raw signal without the use of a filter. Filtering data entails switching between the spatial and frequency domains. This is the distinction between mapping all signals as a function of place and time and mapping the abundance of any individual signal (frequency) against all frequencies.

2.1 The Fourier Series

The Fourier series is a mathematical approach for expressing a periodic function as an infinite sum of sines and cosines. It is based on the assumption that any periodic function can be approximated by a sum of simpler trigonometric functions.

The general form of the Fourier series is given by:

$$f(x) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)]$$

where a_0 is the DC or average value of the function, and a_n and b_n are the Fourier coefficients. These coefficients are given by:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx$$

The Fourier series can also be written in complex form as:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

where c_n is the complex Fourier coefficient given by:

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx$$

The Fourier series allows a periodic function to be represented by a finite number of terms in the series, which can be computed using the Fourier coefficients. The accuracy of the representation depends on the number of terms used in the series, with more terms leading to a more accurate representation. It has numerous applications, including signal analysis and synthesis, image processing, and solving differential equations.

2.2 The Fourier Transform

Fourier transform is a tool to decompose a function from the time domain to the frequency domain. It is an integral defined over the entire line and computed over a definite domain. This tool can be used to transform a sophisticated function into another form and decompose it into a frequency-amplitude image.

When x $\varepsilon[-\infty, \infty]$, the Fourier transform in one dimension is defined as:

$$F(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-ikx}dx$$

And the inverse Fourier transform is defined as:

$$f(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(k)e^{-ikx}dk$$

2.3 The Fast Fourier Transform

To solve this problem, we will apply the Fast Fourier Transform (FFT algorithm) to perform the Fourier transform and inverse Fourier transform. It is widely used because of its low operation count O(NlogN) and good accuracy. In Python, we use functions such as scipy.fft.fft(x), scipy.fft.ifft(x), scipy.fft.fftn(x) to perform forward, backward, one-dimensional and multidimensional FFT.

2.4 Spectral Averaging

We use spectral averaging to recover the clear spectral signature, or center frequency. The target object's central frequency is a specific frequency range among all the noisy signals. The basic principle behind this strategy is that white noise over signals should add up to zero on average. As a result, by averaging a given set of realizations, we can remove the majority of the white noise and obtain a distinct structure of central frequency.

2.5 Spectral Filtering

Spectral filtering is a technique for extracting information at a certain frequency. It may filter out the required frequency by removing the majority of the white noise. Most data in real-world situations will have noise mixed in with the signals we receive. To avoid noise errors, we must first conduct frequency filtering before applying FFT. To eliminate the noise, we utilized the Gaussian filter:

$$F(k) = exp(-\tau(k-k_0)^2$$

3 Numerical Methods

Initially, we load data from Kraken.mat, which contains 49 observations regarding the Kraken's location. The spatial domain and frequency domain (number of frequencies in each direction in the spatial domain) are then defined. In the spatial domain, we partition x, y, and z $\varepsilon[-L, L]$ into n discrete modes. The frequency domain is then rescaled by multiplying it by $\frac{2\pi}{L}$ because the FFT expects a 2π periodic domain. Following that, we shift the frequency domain aces and use meshgrid to create three-dimensional arrays with the X, Y, Z coordinate system and the Kx, Ky, Kz frequency scheme.

In the averaging section, we first flatten the realization data into three dimensions before performing multi-dimensional FFT with scipy.fft.fftn() to add up the frequency. We rescale the gathered frequencies of signals once we get them. Because it may contain complex numbers, we calculate the signal's intensity by taking the absolute value of the cumulative frequencies. Next we look for the strongest signal in the array and use np.unravel index to get its positions.

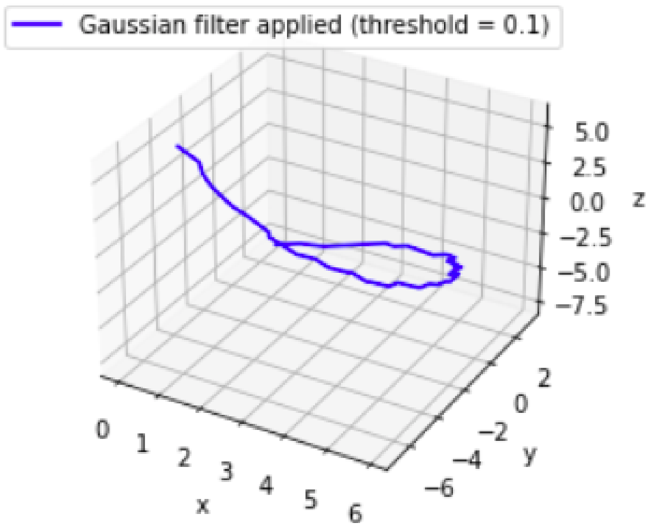
We use the equation (mentioned in the filtering part) to generate a multi-dimensional Gaussian filter for the filtering part. In the averaging stage, we apply the filter by multiplying the data in each direction after rearranging the data for each realization. We take the absolute value of the result to remove the impacts of the complex numbers, and then look for the signal with the highest absolute value. The position of the Kraken can be determined using the coordinates of the maximum signal for each realization. At the end, we save the position data into an array.

Based on the position information we obtained from all the steps above, we can plot out the path of the Kraken by using plt.axes(projection='3d').

4 Results

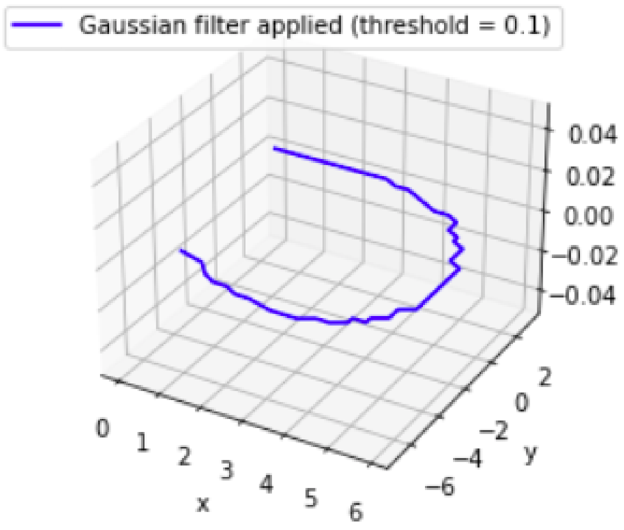
The Kraken location was determined at each time point and a plot of the trajectory can be seen in the Figure 1 below. This figure provides a clear picture of the Kraken's movement over time and can be used to identify any patterns or trends in its behavior.

Figure 1. Kraken's path



Finally, by plotting the (x, y) points of the Kraken's path, the team can identify the optimal points on the ice for a defenceman to make a hard check. This information can be used to develop game strategies and improve the team's chances of winning.

Figure 2. Kraken's path in x-y plane



5 Conclusion

In this problem, the use of averaging and filtering in Fourier space was quite successful in determining the characteristic spatial frequency and then the Kraken's tracks since the estimated trajectory was smooth. The same process might not work well if we are given a more complex system, for example if there were more than one characteristic frequency, the data would require numerous filters to be applied.

6 Acknowledgment

I received some helpful guidance from the TAs throughout this project. They provided useful insights on how to approach certain problems and were always available to answer any questions I had. I appreciate their support and am grateful for the assistance they provided.

7 Appendix

```
import numpy as np
import scipy.fft
import scipy.io as sio
import copy
import scipy
from mpl_toolkits import mplot3d
#%matplotlib inline
import matplotlib.pyplot as plt

data = sio.loadmat('Kraken.mat')
#data
L=10
n=64
x2 = np.linspace(-L, L, n+1)
x = x2[0:n]
y = x
z = x
arr1 = np.arange(0,32)
#print(arr1)
```

```
arr2 = np.arange(-32,0)
#print(arr2)
k = (2*np.pi/(2*L))*np.concatenate((arr1, arr2), axis=0)
#print(k)
#len(k)
ks = scipy.fft.fftshift(k)
#print(ks)
[X,Y,Z] = np.meshgrid(k,k,k)
[Kx, Ky, Kz] = np.meshgrid(ks, ks, ks)
Uk = np.zeros((n,n,n))
#len(data['Kraken'])
data_K = data['Kraken']
data_K.shape
#sum all realizations in frequency space
for j in range(49):
    Un = data_K[:,j].reshape(n, n, n, order = 'F' )
    M = np.amax(abs(Un))
    Uk = Uk + scipy.fft.fftn(Un)

#after your for loop ends, save the sum as variable A1
A1 = Uk
#A1
#Average the sum over the 49 realizations (i.e., A1/49) and save as A2
A2 = A1/49
#find the peak frequencies in x, y, and z directions; i.e., find the
#max in each direction of the normalized sum A2.
#save these variables as A3, A4, and A5
index = np.argmax(Uk)
indexes = np.unravel_index(index, Uk.shape)
#print(indexes)

A4 = X[indexes].copy()
A3 = Y[indexes].copy()
A5 = Z[indexes].copy()

#Simple gaussian filter to apply in frequency domain
gaussian_filter = np.exp(-0.1*((X - X[indexes])**2 + (Y - Y[indexes])**2 +(Z - Z[i
A6 = gaussian_filter.copy()
x_pos = np.zeros(49)
y_pos = np.zeros(49)
z_pos = np.zeros(49)

for t in range(49):
    Un = np.reshape(data_K[:, t], (n, n, n), order = 'F')
    fft_Un = scipy.fft.ifftn((scipy.fft.fftn(Un))*gaussian_filter)
    abs_fft_Un = np.abs(fft_Un)
    max_val = np.argmax(abs_fft_Un)
    #print(max_val)
    indexes1 = np.unravel_index(max_val, Un.shape)
    #print(indexes1)
    x_pos[t] = x[indexes1[0]]
    y_pos[t] = y[indexes1[1]]
    z_pos[t] = z[indexes1[2]]

A7 = x_pos.reshape(1,49)
A8 = y_pos.reshape(1,49)
A9 = z_pos.reshape(1,49)

%matplotlib inline
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot3D(x_pos, y_pos, z_pos, 'b', label = 'Gaussian filter applied (threshold =
#ax.plot3D(X[indexes], Y[indexes], Z[indexes], 'r', label='No filter')
ax.set_title("Figure 1. Kraken's path")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
ax.legend(loc ='upper right');
plt.savefig('CP1_01.png')

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(x_pos, y_pos, 'b', label = 'Gaussian filter applied (threshold = 0.1)')
#ax.plot3D(X[indexes], Y[indexes], Z[indexes], 'r', label='No filter')
ax.set_title("Figure 2. Kraken's path in x-y plane")
ax.set_xlabel("x")
ax.set_ylabel("y")
#ax.set_zlabel("z")
ax.legend(loc ='upper right');
plt.savefig('CP1_02.png')
```