AMATH 582 Homework 1: A Submarine Problem

Emma Shie Nuss, github: emmashie

Abstract

A submarine emitting an unknown frequency was known to be sailing in the Puget Sound. Through the use of Fourier analysis and a spatio-temporal acoustic dataset, the submarine frequency was identified as $\vec{k} = (5.1, -7, 2.1)$. A 3D Gaussian filter was applied to the data in Fourier space and the path of the submarine was identified. From the submarine path, the horizontal coordinates were found to allow a P-8 Orior aircraft to follow the submarine path.

1. Introduction and Overview

Our goal was to locate a submarine sailing around the Puget Sound. This submarine has new technology and emits an unknown acoustic frequency. We have access to 3-dimensional acoustic data over a 24 hour period at half-hour increments. In addition to locating the submarine path, we also aim to determine the flight path of a P-8 Orion aircraft to follow the submarine's path. To achieve our goal, we must: 1) identify the frequency generated by the submarine, 2) clean up the noisy data, 3) identify the location of the submarine over the 24 hour period, and 4) determine the flight path of the aircraft. This report outlines the implementation of Fourier analysis to achieve the aforementioned goal of locating the submarine.

2. Theoretical Background

2.1. Fourier Transform and Series

The Fourier transform (Eq. 1) is a useful mathematical function that transforms a function in space or time and converts not only to frequency or wavenumber space, but breaks the spatial or time-series data into sines and cosines of different frequencies.

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) \mathrm{e}^{\mathrm{i}kx} \mathrm{d}x \tag{1}$$

This relationship is powerful, but assumes an infinite domain $x \in (-\infty, \infty)$. In the case of our submarine problem, we have a finite domain with $x \in [-L, L]$, $y \in [-L, L]$, and $z \in [-L, L]$. We can write the Fourier transform (Eq. 1) as an infinite series where the spatial domain is finite. In 1-D we have,

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kx) + b_k \sin(kx)), \; x \in [-L, L] \tag{2}$$

Multiplying both sides of Eq. 2 by cos(mx) where m is a positive integer and integrating from -L to L, we get that

$$a_k = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{\pi kx}{L}\right)\mathrm{d}x,$$
$$b_k = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{\pi kx}{L}\right)\mathrm{d}x, \tag{3}$$
$$a_0 = \frac{1}{2L}\int_{-L}^{L} f(x)\mathrm{d}x$$

Eq. 2 and 3 provide the ability to decompose a function into an infinite series of cosines and sines of different frequencies, but given that you have a function f(x). In our submarine case, we are given discrete data and do not know what f(x) is and thus need another form to make use of the Fourier transform.

2.2. Discrete Fourier Transform (DFT)

The discrete Fourier transform takes a finite number of equally spaced data from a function and converts the data to equally spaced frequency data. If we have a time-series of N data points $\{x_0, x_1, ..., x_{N-1}\}$, that are equally spaced, then the DFT is given by

$$\hat{x_k} = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}} \tag{4}$$

Instead of infinitely many frequencies, the DFT gives a finite number of frequencies dependent on the number of data points available.

2.3. Fast Fourier Transform

The DFT described above is useful for analyzing time series data; however, computationally is slow to compute. The The Fast Fourier transform (FFT) was developed by Cooley and Tukey in the mid 1960s to speed up the process of calculating DFTs (Kutz, 2013). The FFT method divides a series of N data points, usually powers of two (i.e. $N = 2^n$), into two DFTs of length N/2. This process can be repeated iteratively and is known as the divide and conquer method.

2.4. Filtering

The FFT increases the ability to apply the mathematical concept of Fourier transforms to real world data. One application of the FFT is in signal processing. Our submarine problem is an example of this type of application. Our goal is to identify the path of a submarine emitting a particular frequency. The submarine frequency is not the only frequency being detected by our acoustic instruments. Therefore our data is noisy and we need tools to be able to clearly identify the location of the submarine. Filtering is a useful tool to help eliminate frequency data at frequencies we are not interested in. The Gaussian filter is a simple and useful filter and is defined as

$$F(k) = e^{-\tau(k-k_0)^2}, \tag{5}$$

where k_0 is the frequency you want to isolate, τ is a constant that scales the width of your filter, and k are the range of frequencies you are filtering over. Eq. 5 works by centering a Gaussian function around a frequency of interest, k_0, thus is close of 1 near k_0 and goes to zero as you move away from k_0. Multiplying Eq. 5 with Fourier transformed data isolates the frequency of interest by multiplying other frequencies by small values close to zero.

3. Algorithm Implementation and Development

Taking the knowledge of Fourier transforms and filtering, we can take the acoustic data and:

1. Find the frequency emitted by the submarine
2. Identify the path of the submarine
3. Provide x and y coordinates for the aircraft

Finding the submarine frequency is outlined in Algorithm 1 and identifying the submarine and aircraft paths is outlined in Algorithm 2.

Algorithm 1: Submarine Frequency Algorithm

Import data from `subdata.csv`
Define spatial dimensions L and n
Calculate frequencies (k) scaled by $\frac{\pi}{L}$ and fftshift frequencies to put in order (ks)
Create 3-D meshgrids of x, y, z and kx, ky, kz of size [64, 64, 64] from ks, L, and n
for each time step **do**
  Reshape column of data into 3-D array of [x,y,z]
  Concatenate each 3-D array every timestep to create a 4-D array of [t, x, y, z]
end for
Fourier transform and fftshift the spatial data
Time-average the transformed and shifted data
Normalize the time-averaged and shifted data
Find indices where the normalized data is above a threshold (0.7)
Match indices to kx, ky, kz values
Calculate the average kx, average ky, and average kz to find the center of mass of the frequencies

Algorithm 2: Submarine and Aircraft Path Algorithm

Create 3-D Gaussian filter using our average kx, ky, and kz from 1
for each time step **do**
  Filter Fourier transformed and shifted data by multiplying data with the Gaussian filter
end for
Inverse Fourier transform and shift the transformed data to get filtered acoustic data
Normalize filtered acoustic data
for each time step **do**
  Find indices in normalized filtered acoustic data above threshold (0.7) and average indices to find the center of mass
end for
Match the integer values of the indices to x, y, z locations
Plot submarine path and aircraft path

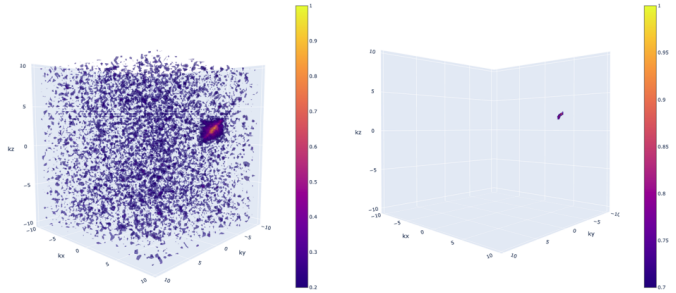

Figure 1: Fourier transformed, time-averaged, and normalized data plotted with threshold 0.2 (left panel) and threshold 0.7 (right panel). Both plots clearly show where the submarine is in frequency space. The right panel demonstrates how effective the threshold is in isolating the area in frequency space that the submarine frequency is.

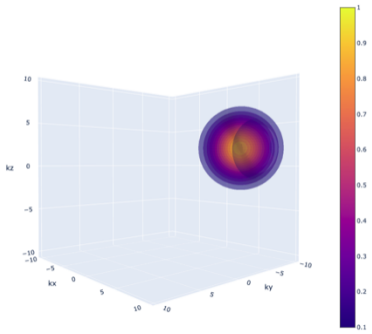

Figure 2: Gaussian filter used to isolate submarine frequencies.

4. Computational Results

4.1. Submarine Frequency

Following the algorithm above (Alg. 1), the submarine frequencies were found through Fourier transforming the data, averaging over the time domain, and normalizing the data. A threshold of 0.7 was used to isolate the submarine frequency and those frequencies were averaged to find the center of the mass in Figure 1 (right panel). The center submarine frequency was found to be $\vec{k} = (5.1, -7, 2.1)$.

4.2. Submarine Path

Taking the frequencies found above, a 3D Gaussian filter was created (Fig. 2) to isolate the submarine frequency and filter out the noise. The Fourier transformed data was filtered at each timestep and then inverse Fourier transformed back to acoustic data. A threshold of 0.7 was used to find the locations of the submarine through time from the filtered acoustic data (Table 1). The submarine's path starts deeper at (2.8, 0, -8.1) and then spirals towards the surface ending at a location of (-5.3, 0.6, 6.2) (Fig. 3).

Time-step	1	5	9	13	17	21	25	29	33	37	41	45	49
X-Coordinate	2.8	3.1	2.8	2.2	0.9	-0.6	-2.5	-4.1	-5.6	-6.6	-7.2	-6.6	-5.3
Y-Coordinate	0.0	1.2	2.8	4.1	5.0	5.6	5.9	5.6	5.3	4.4	3.4	2.2	0.6
Z-Coordinate	-8.1	-6.9	-5.6	-4.7	-3.4	-2.2	-0.9	0.3	1.6	2.5	3.8	5.0	6.2

Table 1: Spatial coordinates (x,y,z) of the submarine at select time steps.

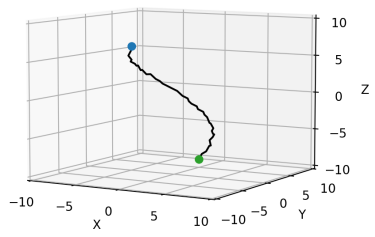

Figure 3: The path of the submarine through time with the green dot denoting the starting location and the blue dot denoting the end location.

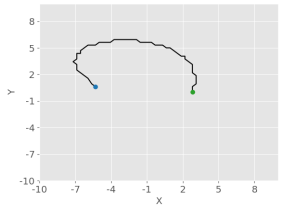

Figure 4: The path of the aircraft following the submarine with the green dot denoting the starting location and the blue dot denoting the end location.

4.3. Aircraft Path

To follow the submarine's path, the aircraft only needs to know the horizontal coordinates of the submarine (i.e. x and y). The submarine's path in the x-y plane appears as a half-circle as it spirals upward (Fig. 4).

5. Summary and Conclusions

The technique of using Fourier transforms, averaging, and filtering is an effective method for finding the path of a submarine from spatial-temporal data. Averaging many realizations of the Fourier transformed data allowed for the submarine frequency to be identified. Filtering helped to isolate the submarine frequency in Fourier space and thus allowed for the submarine to be tracked at each time step though space.

References

J. N. Kutz, Data-driven modeling & scientific computation: methods for complex systems & big data, Oxford University Press, 2013.

Appendix A Python Functions

Relevant Python functions:

- `data = pd.read_csv(filename, header=None)`: Reads in data from a csv file with no header using the Pandas python package
- `x = np.arange(0, n)`: Creates a 1D array of integers starting at 0 up to `n-1`
- `z = np.append(x, y)`: Appends array y to the end of array x and stores it in variable z
- `k = np.fft.fftshift(k)`: Shifts the zero-frequency component to the center of the spectrum
- `[X, Y, Z] = np.meshgrid(x, y, z, indexing='ij')`: Creates a 3D grid based on the coordinates contained in the vectors `x`, `y`, and `z`. Setting `indexing='ij'` ensures that the grid is of size (x, y, z) rather than (y, x, z).
- `D = np.reshape(data, (i, j, k), order='F')`: A column of data is reshaped into size `(i, j, k)`, setting `order='F'` ensures that the reshaping is consistent with Matlab's reshape function and starts with the i dimension rather than the k dimension
- `Dspec = np.fft.fftn(D, axes=[i,j,k])`: The n-dimensional Fourier transform is taken of axes `(i,j,k)`
- `ind = np.argwhere(D == condition) or np.where(D == condition)` Finds the indices of an array where a logical condition is satisfied
- `np.asarray(D)`: Takes a list or some non-array and converts it to an array type
- `L = [i for i in range(n)]`: (List comprehension) Creates a list of `0` to `n-1`, can be used for high dimensions of lists and with if conditional statements
- `os.path.join(path, filename)`: Creates a path object based on the `path` and `filename` given

Appendix B Python Code

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import plotly.graph_objects as go
plt.ion()

# definte data path and filename
datapath = '../data/'
filename = 'subdata.csv'

# load in data and reformat complex numbers for python
subdata = pd.read_csv(os.path.join(datapath, filename), header=None)
subdata = subdata.applymap(lambda s: np.complex(s.replace('i', 'j')))
data = subdata.values

# define spatial and frequency domains
L = 10
n = 64
k_pos = np.arange(0, n/2)
k_neg = np.arange(-n/2, 0)
k = (2*np.pi/(2*L))*np.append(k_pos, k_neg)
ks = np.fft.fftshift(k)

x2 = np.linspace(-L,L,n+1)
x = x2[:n]
y = x
z = x
[X, Y, Z] = np.meshgrid(x,y,z, indexing='ij')
[Kx,Ky,Kz] = np.meshgrid(ks,ks,ks, indexing='ij')

### NOTE: Matlab convention is (Y, X, Z) in meshgrid versus (X, Y, Z) here,
### so x and y are actually flipped throughout the code
### variables are not switched, but x-y labels for plotting and submarine
### coordinate reporting are switched to be consistent with Matlab


# reshape data into 4-d time-spatial data array
Un = np.asarray([np.reshape(data[:,i], (n,n,n), order='F') for i in range(data.shape[-1])])

# find the frequency of the submarine
Un_fft = np.fft.fftshift(np.fft.fftn(Un, axes=[1,2,3]), axes=[1,2,3])
Un_avg = np.mean(Un_fft, axis=0, dtype=complex)
Un_avg_normalized = np.abs(Un_avg)/np.max(np.abs(Un_avg))

threshold = 0.7

ind = np.where(Un_avg_normalized>=threshold)

kx = ks[ind[0]]
ky = ks[ind[1]]
kz = ks[ind[2]]

kx_avg = np.mean(kx)
ky_avg = np.mean(ky)
kz_avg = np.mean(kz)
```

```
if 0: # plotting code for isosurface of normalized and averaged data
        fig = go.Figure(data = go.Isosurface(
                                        x=Kx.flatten(),
                                        y=Ky.flatten(),
                                        z=Kz.flatten(),
                                        value=Un_avg_normalized.flatten(),
                                        isomin=0.2,
                                    isomax=1,
                                    surface_count=15,
                                    opacity=0.3))
        fig.update_layout(scene = dict(
                    xaxis_title='ky',
                    yaxis_title='kx',
                    zaxis_title='kz'))
        fig.show()

kx_centered = (Kx - kx_avg)**2
ky_centered = (Ky - ky_avg)**2
kz_centered = (Kz - kz_avg)**2

# creating gaussian filter
tau = 0.1

gaussian_filter = np.exp(-tau * kx_centered)*np.exp(-tau * ky_centered)*np.exp(-tau * kz_centered)

if 0: # plotting gaussian filter
        fig = go.Figure(data = go.Isosurface(
                                        x=Kx.flatten(),
                                        y=Ky.flatten(),
                                        z=Kz.flatten(),
                                        value=gaussian_filter.flatten(),
                                        isomin=0.1,
                                    isomax=1,
                                    surface_count=15,
                                    opacity=0.3))
        fig.update_layout(scene = dict(
                    xaxis_title='ky',
                    yaxis_title='kx',
                    zaxis_title='kz'))
        fig.show()

# filtering the data with a gaussian filter at the frequency we found prior
Un_fft_filtered = np.asarray([Un_fft[i,:,:,:]*gaussian_filter for i in range(len(Un))])

Un_filtered = np.fft.ifftn(np.fft.ifftshift(Un_fft_filtered, axes=[1,2,3]), axes=[1,2,3])
Un_filtered_normalized = np.asarray([np.abs(Un_filtered[i,:,:,:])/np.max(np.abs(Un_filtered[i,:,:,:]))
                    for i in range(len(Un_filtered))])

# find locations at each time step that have signal above threshold
sub_ind = np.asarray([np.mean(np.asarray(np.argwhere(Un_filtered_normalized[i,:,:,:]>=threshold)), axis=0)
                    for i in range(len(Un_filtered_normalized))])

# finding the center of mass of the signal
sub_x = np.asarray([x[int(sub_ind[i,0])] for i in range(len(sub_ind))])
sub_y = np.asarray([y[int(sub_ind[i,1])] for i in range(len(sub_ind))])
sub_z = np.asarray([z[int(sub_ind[i,2])] for i in range(len(sub_ind))])
```

```
if 0: # plotting submarine path
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.plot(sub_y, sub_x, sub_z, color='black')
        ax.plot(sub_y[-1], sub_x[-1], sub_z[-1], 'o', color='tab:blue')
        ax.plot(sub_y[0], sub_x[0], sub_z[0], 'o', color='tab:green')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.set_xlim((-L,L))
        ax.set_ylim((-L,L))
        ax.set_zlim((-L,L))
        ax.locator_params(nbins=5)
        fig.savefig('sub_path.png')

if 0: # plotting aircraft path
        plt.style.use('ggplot')
        fig, ax = plt.subplots()
        ax.plot(sub_y, sub_x, color='black')
        ax.plot(sub_y[-1], sub_x[-1], 'o', color='tab:blue')
        ax.plot(sub_y[0], sub_x[0], 'o', color='tab:green')
        ax.set_xlabel('X', fontsize=14)
        ax.set_ylabel('Y', fontsize=14)
        ax.set_xticks(np.arange(-L, L)[::3])
        ax.set_yticks(np.arange(-L, L)[::3])
        ax.set_xticklabels(np.arange(-L, L)[::3], fontsize=14)
        ax.set_yticklabels(np.arange(-L, L)[::3], fontsize=14)
        ax.set_xlim((-L,L))
        ax.set_ylim((-L,L))
        fig.savefig('air_path.png')

if 0:
        # write x, y, z coordinates to csv for table generation
        f = open('xyz_coords.csv', 'w')
        subx = sub_x[::4]
        suby = sub_y[::4]
        subz = sub_z[::4]
        for i in range(len(subx)-1):
                f.write("%0.1f," % (subx[i]))
        f.write("%0.1f\n" % (subx[-1]))
        for i in range(len(suby)-1):
                f.write("%0.1f," % (suby[i]))
        f.write("%0.1f\n" % (suby[-1]))
        for i in range(len(subz)-1):
                f.write("%0.1f," % (subz[i]))
        f.write("%0.1f\n" % (subz[-1]))
        f.close()
```