AMATH 482 Homework 1

Zach Zlepper

January 24, 2020

Abstract

After sending out a signal to detect an enemy submarine, we get back a signal with a lot of noise woven into it. In the following paper, I will outline a way to remove this noise using Fourier Transforms with averaging and filtering. We expect to better track the submarine after processing our signal. After averaging out the white noise and using a 3 dimensional Gaussian filter, we succeeded in finding the specific coordinates of the underwater submarine.

1 Introduction and Overview

In the deep ocean, we are commanding a submarine that is in a battle with another enemy sub. We have just sent out 49 pings to see if we can detect where the enemy subs location is. However, because of random noise in our environment, we receive a jumbled mess.

In order to remove the noise from our data, we must first average over our 49 data points, taken at 30 minute intervals during a 24 hour time span, in Fourier Space. This technique will remove excess white noise. Once we do this, we can find the largest frequency and hone in on it. Using a Gaussian filter, we can set unimportant frequencies to zero and then Inverse Fourier Transform the data back into signal space to find the location of the enemy sub. Then we can launch a torpedo and win the battle.

2 Theoretical Background

We start with the question of how to break down signals into certain frequencies. Fourier introduced the concept of representing a given function $f(x)$ with a trigonometric series of sines and cosines. To start, we will see that there is an algorithm called Fourier Transform that converts a function $f(x)$ with $x \in \mathbb{R}$ into $\hat{f}(k)$ where k is a certain frequency.

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

If we have a bunch of frequencies and want to go back to the signal, there is an algorithm called the Inverse Fourier Transform.

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

The downfall of Fourier Transforms are the infinite domain. For most applications, data is finite. So, we will introduce a finite form called the Fourier Series. Let L be the size of the spacial domain.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n cos(\frac{nx\pi}{L}) + b_n sin(\frac{nx\pi}{L}) \right) \quad x \in [-L, L]. \tag{3}$$

We can solve for the Fourier Coefficients a_0, a_n, and b_n. These coefficients act as weights to the frequencies for each frequency n. To calculate the coefficients we use the following formulas:

$$a_n = \frac{1}{L} \int_{-L}^{L} f(x) cos(\frac{nx\pi}{L}) dx \tag{4}$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x) sin(\frac{nx\pi}{L})dx \tag{5}$$

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)dx \tag{6}$$

Notes:

1. Since sines and cosines are periodic, f(x) must be periodic.
2. For continuous f(x) , the Fourier Series converges to the value at f(x) for each x in the domain. For discontinuous functions, the Fourier series converges to the average of the left and right values at x.

Here, we see we still have a problem for discrete data such as a sound bit. We have solved the issue of the infinite domain, however, we must change this equation to accommodate discrete data. This finite domain for discrete data can be captured in the Discrete Fourier Transform (DFT).

$$\hat{x}_k = \frac{1}{N}\sum_{i=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}} \tag{7}$$

A restriction for the Discrete Fourier Transform is that we have finitely many frequencies we can detect as well. $k = 0, 1, 2, ..., N-1$. With the DFT, we run into a problem called aliasing. The k and k+N frequencies are the same because higher frequencies are lost between sample points and look the same as . lower frequencies.

Finally, we the Heisenberg Uncertainty Principle makes use trade off between a lot of information about time and a lot of information about the frequencies. We must balance these two phenomena to accomplish our task of finding the submarine.

3 Algorithm Implementation and Development

Largely, we are going to use the DFT to move in between signal and frequency space. However, DFT is that it is slow $O(N^2)$. There is a faster algorithm called the Fast Fourier Transform that has a finite domain, discrete points, and runs in $O(NLog(N))$ time. Typically, we want to use a power of 2 as the number of data points. This allows the FFT algorithm to run more efficiently. The FFT is uses the DFT but breaks down the problem recursively. There are two components to our algorithm which are averaging and filtering. Before we can start our algorithms, we must set up our parameters. We must,

1. Define our spacial and Fourier spaces and grids using commands such as `meshgrid`.
2. Re-scale our domain to fit a 2π periodic signal since maltab works in 2π and we are working in 2L.

Averaging is a technique to remove white noise. Since white noise is random and centered around a mean of 0, averaging reduces white noise. This helps us find our maximum frequency to focus on for filtering in a later step.

Algorithm 1: Averaging

```
Import data from subdata.mat
for $j = 1 : 49$ do
    1. Extract measurement $j$ from subdata.mat
    2. Reshape data into a 64x64x64 matrix
    3. FFT the data
    4. Add the frequencies to a cumulative variable
end for
fftshift, absolute value the data, and then divide by 49 to find average frequency.
Then, pull the largest frequency from all of the averaged frequencies.
```

Now that we have found the largest frequency, we are going to build a 3-D Gaussian filter by multiplying 3 1-D Gaussians together. Let x_0, y_0,and z_0 describe the center of the Gaussian and τ be a parameter that controls the spread of the Gaussian.

$$F(k) = e^{-\tau(kx-x_o)^2+\tau(ky-y_o)^2+\tau(kz-z_o)^2} \tag{8}$$

By multiplying the frequencies by the gaussian filter, we send most frequencies to zero so we can better focus on the frequencies that matter. We are also going to center the 3-D filter around the coordinates that describe the maximum frequency which we found by averaging.

Algorithm 2: Filtering

```
Import data from subdata.mat
for $j = 1 : 49$ do
    1. Extract measurement $j$ from subdata.mat
    2. Reshape data into a 64x64x64 matrix
    3. FFT the data
    4. fftshift the data
    5. Multiply the data and the 3-D gaussian filter together
    6. Then ifft back into signal space
    7. Find the maximum signal and mark the coordinates.
end for
Plot the coordinates to see the enemy submarines location
```

4 Computational Results

After averaging over the 49 data realizations, using the algorithm described in Algorithm 1, we can pull out our maximum frequency to filter around. Then, we take this central frequency and perform Algorithm 2 on of Fourier Transformed data. After, we can bring our data back into signal space which yields a coordinate system. This can be referenced by Table 1 or seen in Figure 1.

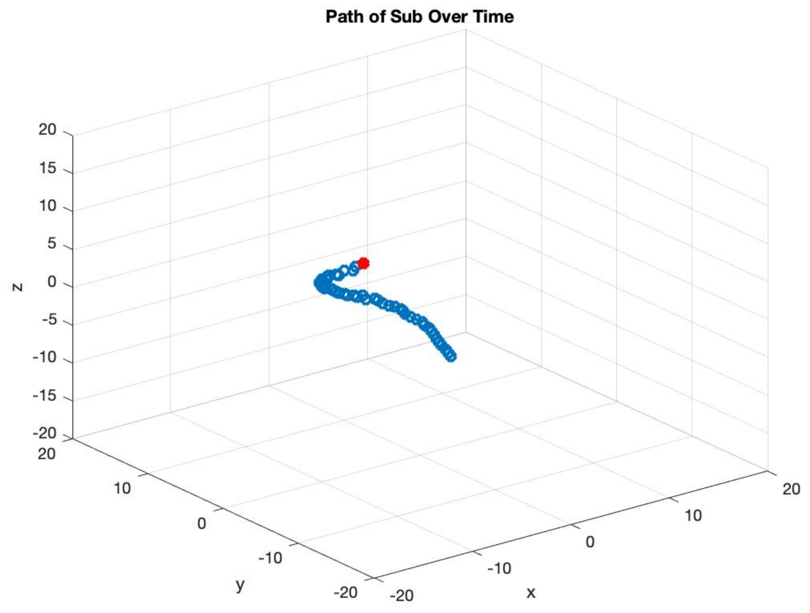

Figure 1: Submarine Graph

Measurement	X Position	Y Position
1	3.1250	0
2	3.1250	0.3125
3	3.1250	0.6250
4	3.1250	1.2500
5	3.1250	1.5625
6	3.1250	1.8750
7	3.1250	2.1875
8	3.1250	2.5000
9	3.1250	2.8125
10	2.8125	3.1250
11	2.8125	3.4275
12	2.5000	3.7500
13	2.1875	4.0625
14	1.8750	4.3750
15	1.8750	4.6875
16	1.5625	5.000
17	1.2500	5.000
18	0.6250	5.3125
19	0.3125	5.3125
20	0	5.630
21	-0.6250	5.6250
22	-0.9375	5.9375
23	-1.25	5.9375
24	-1.8750	5.9375
25	-2.1875	5.9375
26	-2.8125	5.9375
27	-3.1250	5.9375
28	-3.4375	5.9375
29	-4.0625	5.9375
30	-4.3750	5.9375
31	-4.6875	5.6250
32	-5.3125	5.6250
33	-5.6250	5.3125
34	-5.9375	5.3125
35	-5.9375	5.0000
36	-6.2500	5.0000
37	-6.5625	4.6875
38	-6.5625	4.3750
39	-6.8750	4.0625
40	-6.8750	3.7500
41	-6.8750	3.4375
42	-6.8750	3.4375
43	-6.8750	2.8125
44	-6.5625	2.5000
45	-6.2500	2.1875
46	-6.2500	1.8750
47	-5.9375	1.5625
48	-5.3125	1.2500
49	-5.0000	0.9375

Table 1: Submarine Location Coordinates.

5 Summary and Conclusions

This paper has demonstrated the use of Fast Fourier Transforms, averaging, and filtering to denoise data. By using these techniques, we found the enemy submarine. But, these techniques can be braodly applied to many signal processing applications.

Appendix A

MATLAB functions used and implementation

abs(X) : Returns the absolute value of every element in X, or complex magnitude if the element is complex.

$[i, j, k]$ =ind2sub(size,index) : Takes linear indices and converts to "i", "j", and "k" which are coordinates in a cube.

fftn(X) : Performs a N-D Fast Fourier Transform on X.

fftshift(X): Places the the zero-frequency components of X and the output of fftn to the center of the array.

ifftn(X) : Performs a N-D Inverse Fast Fourier Transform on X.

ifftshift(X) : Performs the inverse of fftshift(X).

isosurface(X,Y,Z,V,isovalue) : Plots the isosurface from volume data V on grid of X, Y, Z.

linspace(x1,x2,n) : It generates a linearly spaced vector with n evenly space points

$[X, Y, Z]$ = meshgrid(x,y,z) : Creates 3-D grid coordinates defined by the X, Y, and Z vectors.

max(A) : Returns the maximum value in vector A.

$[M, I]$ = max(A) : Returns the maximum value M and the index I of the matrix.

plot3(X1,Y1,Z1) : Displays a 3-D plot with coordinates X1, Y1, Z1

reshape(A,dim) : Reshapes the array A to dimensions dim.

zeros(A,B,C) : Initializes matrix of size A, B, C with zeros.

Appendix B

MATLAB Code

```
clear all; close all; clc;
load subdata.mat % Imports the data as the 262144x49 (space by time) matrix called subdata
L = 10; % spatial domain
n = 64; % Fourier modes
x2 = linspace(-L,L,n+1);
x = x2(1:n);
y =x;
z = x;
k = (2*pi/(2*L))*[0:(n/2 - 1) -n/2:-1];
ks = fftshift(k);
[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

%Pt1 - averaging

Ufave= zeros(n, n, n);
for j=1:49
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    Uf = fftn(Un); %Frequency space
    Ufave = Ufave + Uf;% add up all Frequencies
 end
 Ufave= abs(fftshift(Ufave)/49); % Averages frequency space


 [maximum_val, I] = max(Ufave(:)); %Finds the maximum frequency in Frequency space
 [xI, yI, zI] = ind2sub(size(Ufave),I); %Locates the maximum Frequency placement
 x0 = Kx(xI,yI,zI);
 y0 = Ky(xI,yI,zI);
 z0 = Kz(xI,yI,zI);


 filter = exp(-((Kx - x0).^2 + (Ky - y0).^2 + (Kz - z0).^2));
 % Builds 3-D filter around Location of largest frequency
 coord = zeros(49 ,3);

 for j = 1:49
     Un(:,:,:)=reshape(subdata(:,j),n,n,n);
     Uf = fftn(Un);
     Uf = fftshift(Uf);
     Uf_filtered =  filter.* Uf; %Filters data in frequency space
     Unif = ifftn(Uf_filtered);
     [maximum_val, I] = max(abs(Unif(:))); %Finds locaiton of the largest signal
     [xI, yI, zI] = ind2sub(size(Unif),I);
     coord(j,1) = X(xI,yI,zI);
     coord(j,2) = Y(xI,yI,zI);
     coord(j,3) = Z(xI,yI,zI);
 end

plot3(coord(:,1),coord(:,2),coord(:,3),'o','Linewidth',2);
grid on
```

```
title('Path of Sub over time')
xlabel('x');
ylabel('y');
zlabel('z');
axis([-20 20 -20 20 -20 20])
hold on
plot3(coord(end,1),coord(end,2),coord(end,3),'ro','MarkerFaceColor','r','Linewidth',2);

final = [coord(end,1),coord(end,2),coord(end,3)];
```