AMATH 482 Homework 1:
A Submarine Problem

Malikah Nathani
January 27th, 2021

Abstract

A submarine problem asks to determine the path of a submarine given noisy acoustic frequencies over a 24-hour period in half-hour increments. By using the Fourier Transform and techniques such as averaging and data filtering we can quickly denoise the data and calculate the (x, y, z) coordinates of the submarine over the time period.

1 Introduction and Overview

The area of signal processing has an extensive number of applications, including audio and speech, digital image, and financial signal processing. Many of these fields use the Fourier Transform to break a signal down to its frequencies, then use techniques such as averaging and filtering to uncover specific and important data used to solve a wide array of problems. In the context of this paper, we will apply this method to the field of acoustic wave processing.

In the Puget Sound Area, there is a submarine releasing an unknown acoustic frequency that can be detected. We have recorded these acoustics for a 24-hour time window in half-hour increments. The issue with the data is that it is incredibly noisy, as it has also captured white noise, such as other energies present or other underwater phenomena. With this, it is hard to determine what is the true frequency the submarine released, therefore impossible to plot its path. However, with the use of the techniques mentioned above, we can denoise the acoustic data over the 49 time points to determine the trajectory of the submarine.

In Section 2, I will go about explaining each of the methods used in more detail while section 3 will go over the implementation of these methods in the algorithm used. Sections 4 and 5 will go over the computed results and my conclusions respectively. Finally, I will also present the MATLAB functions used and provide my code.

2 Theoretical Background

2.1 Fourier Transform

When given a certain signal and your goal is to break it down into its different frequencies, one way to do this is to use a *Fourier Transform* $\hat{f}(k)$ written in the formula:

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

The inverse Fourier transform is for when you are given $\hat{f}(k)$ and wish to transform it back to $f(x)$:

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

Using the Fourier Transform can serve a multitude of purposes; however, it is important to note that the wider a function is, the thinner it's Fourier transform is, and vice versa. This is known as the *Heisenberg Uncertainty Principle*. In context to signal processing, this principle applies to knowing time vs frequency information about a signal. The more known about one aspect, the less is known about the other. One assumption in using this transformation is that you have an infinite domain. Similarly, in this problem, we are given discrete data points over a finite interval therefore we use a rendition of the Fourier transform that works for our dataset and time domain.

2.1.1 Discrete Fourier Transform (DFT)

The DFT is a method to determine the Fourier Transform of a function over a finite interval at given discrete data points. Given a sequence of N values that are sampled at equally-spaced points. The DFT will give a sequence of numbers as:

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

One thing to note, since we are given a set of points, we cannot tell if there is an extremely high frequency occurring in between two points. Therefore, the DFT is cut at a maximum frequency to account for this issue. Another aspect to mention is that the DFT uses a total complexity of $O(N^2)$. This can be highly inefficient when working with large amounts of data. Therefore, we implement a Fast Fourier transform to quicken the time spent doing the transformation. Another aspect to note for N, people typically use the powers of 2 for N (i.e. $N = 2^n$).

2.1.2 Fast Fourier Transform (FFT)

The FFT is a method to determine the Fourier Transform of a function at a given set of discrete points. The FFT uses a total complexity of $O(N(\log(N))$ as it implements the divide and conquer algorithm. Given a sequence of length N, the FFT splits the DFT giving two DFTs with length N/2. Repeating this process over and over again will ultimately give the transformation of these values.

One aspect to consider when implementing in MATLAB, is that the FFT assumes 2π periodic signals ($-\pi$, π). Since our frequencies are on the scale of (-L, L), before using the FFT functions in MATLAB it is important to rescale all the frequencies by $\frac{2\pi}{L}$.

2.2 Averaging

White noise can be modelled in data by adding to each Fourier component a normally distributed random variable with zero unit variance and mean. By averaging over all the realizations in the frequency domain, ultimately, the white noise from the realizations will cancel out thus allowing you to find the central frequency, this will be the point where the absolute value of the averaged frequencies is the greatest. By getting the central frequency, we can now understand where to filter the data at every realization, to understand the path of the object.

Although the averaging method is incredibly powerful, there are drawbacks to this method. Since we are averaging over the frequency domain, the white noise is impacting each frequency therefore we cannot average in the time domain, we lose the information when the signal occurs in the time domain. This goes back to the teachings of the Heisenberg uncertainty principle.

2.3 Filtering

Filtering is an incredibly powerful tool to filter out the noise and detect a signal when the central frequency of the signal is known. A filter is a function that you multiply to the signal in the frequency domain to remove undesired frequencies and noise. In filtering at every realization, you are able to easily determine the path of the object. There are many ways to filter a signal, in this problem we are using a Gaussian filter:

$$F(k) = e^{-\tau((kx-kx0)^2+(ky-ky0)^2+(kz-kz0)^2)}$$

Where τ represents the width of the filter while *kx0*, *ky0* and *kz0* represent the center of the filter.

3 Algorithm Implementation and Development

This section details the implementation of the methods above to solve this problem and any other steps taken to compute the submarine path in MATLAB. For the actual MATLAB code, see Appendix B.

The first step is to load your data into MATLAB, then you define your spatial domain L (length of domain) and the number of Fourier modes n, in this case we used n = 2^6. Afterwards, we create our discrete domain and the x, y, and z vectors for the first n points. As mentioned above, it is important to rescale the frequencies by $\frac{2\pi}{L}$ to fit MATLAB domain and then shift those values such that the zero-frequency component is at the center of the array. We also need to convert our data into a 3-D matrix representing our Cartesian coordinates over 49 time realizations, thus we obtain a 64 x 64 x 64 matrix over a fourth dimension of time.

As mentioned above, since the white noise has zero mean, by averaging over all the 49 realizations, the noise will cancel out giving a better idea of the true signal. Again, this needs to be completed in the frequency domain and not time, because the white noise is added onto each frequency individually in the space. In the time domain, the noise does not impact discrete points therefore, averaging over the time realizations will not work. Since it is difficult to see the maximum frequency in this 3-D space, we use built in MATLAB functions (described in detail Appendix A) to find the index of the maximum frequency to obtain the k values in the x, y, and z direction, to get our central frequency. This will allow us to filter at this central frequency over the 49 realizations of time to determine the path of the submarine.

Finally, we use the k values found above to create a Gaussian filter with a τ value of 0.5. τ determines the width of the filter and 0.5. The higher the τ value, the wider the filter thus the more information along the sides of the center frequency is let it. You want a width where not too much information is coming in, as then it may contain all the noise still present in the signal. However, a filter width too small is also undesirable, as that may not contain enough information to give you a proper understanding of the data and what it represents. When determining the τ value needed, you multiply the filter to the signal in the frequency domain and then convert it back into the time domain. This will allow you to correctly visualize the trajectory of the submarine. To find the x, y, and z coordinates, after converting the signal in the time domain, find the index where the signal is at its peak (maximum). You take the values of the index in your 3-D matrix and plot that to get where the submarine is at that time realization. Doing this over the 49 realizations will give you the submarine's path.

4 Computational Results

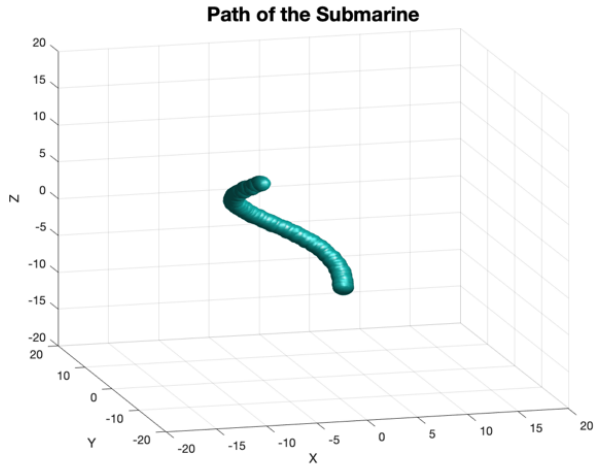

Figure 1: Visualization of the Submarine's path using Isosurface

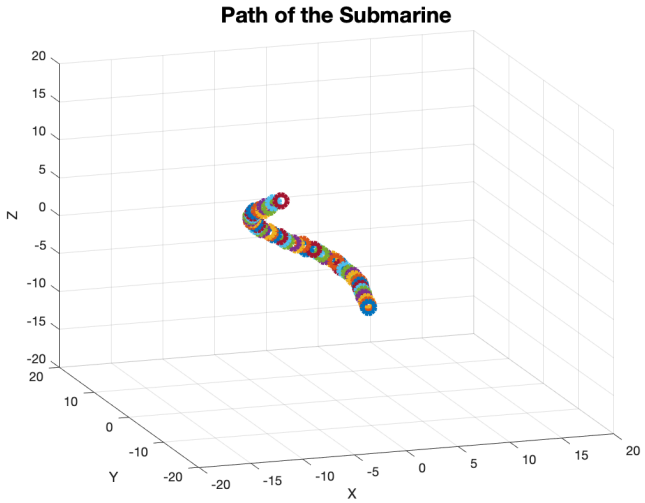

Figure 2: Visualization of the Submarine's path using plot3 command

Time (Hour)													
	0	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**
x	3.1250	3.1250	3.1250	2.1875	1.2500	-0.3125	-2.1875	-4.0625	-5.6250	-6.5625	-6.8750	-6.2500	-5
y	0	1.5625	2.8125	4.0625	5	5.6250	5.9375	5.9375	5.3125	4.6875	3.4375	2.1875	0.9375

Table 1: x, y coordinates of the submarine every two hours in the 24-hour interval

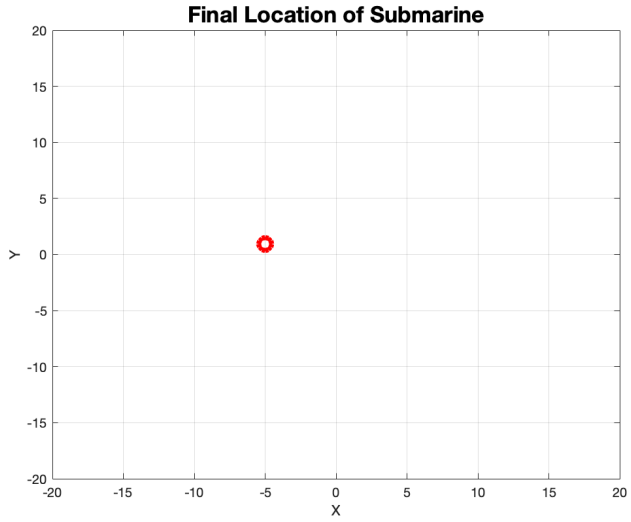

Figure 3: Final location of the Submarine from a bird's eye view (x, y coordinates of submarine)

5 Summary and Conclusions

Through the use of the Fast Fourier transform, averaging, and Gaussian filtering, we were able to denoise the data of the submarine and find its path for each half hour in the 24-hour period. All of these processes serve a different yet incredibly important purpose to solve not only this problem, but many in the realm of signal processing and other data analysis issues. Therefore, the applications of this method are limitless and are worth studying further. The final point of the submarine is (-5, 0.9375).

Appendix A. MATLAB functions used

- `load`: loads the data from the filename specified
- `y = linspace(x1, x2, n)`: generates a row vector of `n` evenly spaced points between `x1` and `x2`
- `Y = fftshift(X)`: rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array
- `[X, Y, Z] = meshgrid(x, y, z)`: returns 3-D grid coordinates defined by the vectors `x`, `y`, and `z`. The grid represented by `X`, `Y`, and `Z` has length(`y`)-by-length(`x`)-by-length(`z`)
- `B = reshape(A, sz)`: reshapes `A` using the size vector `sz` to define the size of `B`

- `Y = fftn(X)`: returns the multidimensional Fourier transform of an N-D array using a Fast Fourier transform algorithm.
- `Y = abs(X)`: returns the absolute value of each element in array `X`. If `X` is complex, returns the complex magnitude.
- `[M, I] = max(X)`: returns the max value as well as the corresponding index in the operating dimension.
- `[row, col] = ind2sub(sz, I)`: returns the arrays `row` and `col` containing the equivalent row and column subscripts corresponding to the linear indices `I` for a matrix of size `sz`.
- `y = exp(x)`: returns the exponential for each element in array `x`.
- `X = ifftn(Y)`: returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm.
- `plot3(X, Y, Z)`: plots coordinated in 3-D space
- `isosurface(X, Y, Z, V, isovalue)`: computes isosurface data from the volume data `V` at the isosurface value specified in `isovalue` for the coordinate arrays `X`, `Y`, `Z`.

Appendix B. MATLAB codes

```
load subdata.mat

L = 10;
n = 64;

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); % frequency of data in x,y,z direction

%% Averaging accross the spectrum to find center frequency
tot = 0;
for j = 1:49
    Un(:,:,:) = reshape(subdata(:,j),n,n,n);
    Unt = fftn(Un);
    tot = tot + Unt;
end

avg = abs(fftshift(tot))/49;

[argval, argmax] = max(abs(avg(:)));
[x1,y1,z1] = ind2sub(size(avg),argmax); % k values for center frequency, need
to shift
```

```
%% Filter data around center frequency
tau = 0.5;
kx0 = Kx(x1,y1,z1);
ky0 = Ky(x1,y1,z1);
kz0 = Kz(x1,y1,z1);

filter = exp(-tau * ((Kx - kx0).^2 + (Ky - ky0).^2 + (Kz - kz0).^2));

%% Determine path for submarine + plot
for j = 1:49
    Un(:,:,:) = reshape(subdata(:,j),n,n,n);

    Unft = filter.* fftshift(fftn((Un)));
    Unf = ifftn(Unft);

    % Plot isosurface
    M = max(abs(Unf(:)),[],'all');
    isosurface(X,Y,Z,abs(Unf)/M,0.7)
    axis([-20 20 -20 20 -20 20]), grid on, drawnow

    % Plot plot3
    [val, m] = max(abs(Unf(:)));
    [xf,yf,zf] = ind2sub(size(Unf),m);

    xfi = X(xf,yf,zf);
    yfi = Y(xf,yf,zf);
    zfi = Z(xf,yf,zf);

    plot3(xfi,yfi,zfi,'o','LineWidth', 5, 'MarkerSize', 10); hold on;
    axis([-20 20 -20 20 -20 20]), grid on, drawnow

    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    title('Path of the Submarine', 'FontSize', 17);
end

% Plot final point
plot(xfi,yfi, 'o', 'LineWidth', 5, 'MarkerSize', 10, 'color', 'r');
axis([-20 20 -20 20]), grid on, drawnow
xlabel('X');
ylabel('Y');
title('Final Location of Submarine', 'FontSize', 17);
```