AMATH 482 Homework 1

Haley Riggs

January 20, 2020

Abstract

The fast Fourier Transform is an algorithm that has numerous applications in applied mathematics. Here, we consider its use in processing acoustic data gathered from a submarine. Once in the frequency domain, the data is averaged to eliminate noise and find the submarine's frequency signature. It is then filtered, using a Gaussian filter function, to determine the submarine's path and find coordinates that could be given to a subtracking aircraft.

1 Introduction and Overview

In this assignment, we used three-dimensional acoustic data to track the current location and trajectory of a submarine in the Puget Sound. New technology allowed the submarine to emit an acoustic frequency, which was recorded every half an hour across a span of 24-hours. The data collected is comparable to a three-dimensional sound map; every point in space has its own amplitude at a given point in time. Together, it gives us the information we need to determine the frequency signature, location, and trajectory of the submarine.

In the report that follows, we use this data to gather information about the submarine. The spectrum of frequencies given by the collected data was averaged to find the frequency signature of the submarine. Then, the data was denoised and filtered about this center frequency point to find the submarine's path. Finally, the coordinates of the submarine's path were used to determine what the path of a P-8 Poseidon aircraft sent to track the submarine should be.

2 Theoretical Background

To track the submarine, we must detect the frequencies that make up its signal. If these frequencies are presented in the form of a function, we want to break it apart into sines and cosines that have different frequencies. This leads us to the Fourier Transform which, for a function $f(x) \in \mathbb{R}$, is defined as:

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

The inverse Fourier transform, which finds $f(x)$ for a given $\hat{f}(k)$, is equal to:

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

Remembering that Euler's formula tells us that $\mathrm{e}^{\mathrm{i}\pi} = \cos\pi + \mathrm{i}\sin\pi$, we may observe that $\mathrm{e}^{\mathrm{i}kx} = \cos kx + \mathrm{i}\sin kx$ and therefore, in the context of the Fourier transform, k provides the frequencies of the waves produced by sine and cosine. It then follows that, from a function of time or space, the Fourier transform produces a function of frequencies. For practical purposes, though, we must consider a form of the Fourier transform that does not require an infinite domain.

This leads us to the Fourier series which, for a periodic function $f(x)$, is defined as

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_n \cos kx + b_n \sin kx) \quad x \in [-\pi, \pi]. \tag{2}$$

Certain formulas may be used to derive the coefficients of the Fourier series. However, the Fourier series requires a concrete function to convert into a series. Since the submarine data is just that, data, and not a function, we cannot use this method either. We require a transform that allows an input of a discrete set of points.

Here we arrive at the Discrete Fourier Transform (DFT). We must note that since it is not really possible to gather information about very high frequency events occurring *between* these points, we can't gather useful information for a very large value of k. The DFT addresses this; it is a version of the Fourier series, truncated at a maximum frequency point. Considering a vector of N data values that are gathered at equally-spaced points given by $\{x_0, x_1, x_2, \ldots, x_N, x_{N-1}\}$, the discrete Fourier transform is equal to:

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n \exp(\frac{2\pi \mathrm{i} k n}{N}). \tag{3}$$

Note that we only consider the frequencies corresponding to $k = 0, 1, 2, \ldots, N-1$.

At this point, we encounter another difficulty; in the DFT, frequencies $-k$ and $k+N$ are the same. It then follows that, as we examine a signal solely at discrete points, we could potentially mistake our signal for a different one. This phenomenon is called *aliasing.*

Instead, as MATLAB does, we may look at frequencies resulting from $k = -N/2, -N/2+1, \ldots, -1, 0, 1, 2, \ldots, N/2-1$. MATLAB gives the resulting values in this order:

$$\{\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N/2-1}, \hat{x}_{-N/2}, \hat{x}_{-N/2+1}, \ldots, \hat{x}_{-1}\} \tag{4}$$

The Fast Fourier Transform (FFT) is basically a faster way of doing the DFT. While the DFT has complexity $O(N^2)$, the FFT has complexity $O(N \log(N))$. Essentially, the FFT algorithm takes the DFT of a sequence that has N values and splits it into two DFTs with $N/2$ values. This is then repeated, resulting in what is called a *divide and conquer algorithm.* It is a good idea to use N values that are powers of 2.

This process can be executed with MATLAB's `fft()` function and, for inverse transforms, `ifft()` function. For higher dimensional Fourier transforms, MATLAB functions `fft2()` and `fftn()`, as well as `ifft2()` and `ifftn()`, do the job. As aforementioned, MATLAB outputs the DFT values (the results of the `fft()` function) in an interesting order. To shift them back to the order we originally had, we must use the `fftshift()` function. When a function has gone through the `fft()`, we say that it is in the frequency domain. Otherwise, we say that it is in the time domain.

One thing to note is that the width of a function and the width of its Fourier transform have an inverse relationship (i.e. a wider function corresponds to a thinner Fourier transform and a thinner function corresponds to a wider Fourier transform). This phenomenon is known as the *Heisenberg uncertainty principle.*

As demonstrated in the submarine data, signals are often noisy in the real world. In general, though, this noise affects all parts of the signal in the same way; it is *white noise.* One of our goals then becomes to *filter* out the noise from the data.

If we know the frequency of the signal, we know that all other frequencies are most likely only noise. We then want to define a filter around this point to remove the extraneous frequencies. In terms of the mathematics of this operation, a filter is simply a function that the signal, which is in the frequency domain, will be multiplied by. We could, for example, choose to use a Gaussian function, where τ corresponds to the width of the filter and k_0 gives the center of the filter, as follows:

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

Filtering is a useful technique but it ignores the fact that the noise is white and that the signal will likely continue to produce data. White noise has zero mean and therefore, if we average the data (which is in *frequency space*) over multiple realizations, the noise will cancel out, producing a result similar to the actual signal. This concept is known as *averaging.*

Just as one might think, the more realizations the better. Also note that it does not matter if the signal comes from a stationary location or not (as in the case of the submarine). The only difference between the two is that for a moving signal, while the shape of the graph in the time domain stays the same, the spike's location moves. Regardless of the location of the spike, as long as the overall shape of the graph is the same, the Fourier transform of it will be the same. This is because the Fourier transform is a global

function, requiring information about the entire domain of the function. It also splits a function apart into its different frequencies. These frequencies may shift but their values won't change with time, resulting in the same Fourier transform regardless of the signal's location. This is known as *translational invariance*. Averaging paints a very good picture of the signal in frequency space but in doing so, erases all information about the location of the signal in the time domain.

3 Algorithm Implementation and Development

In this project, the frequency signature, trajectory, and location of the submarine were discovered. Coordinates for the path of a subtracking aircraft were also discovered. To start, the submarine's acoustic data was loaded into MATLAB. It was of size 262144 by 49, which represented space versus time. Then, the stage was set for the remainder of the algorithm.

The computational domain was defined to be [-10,10] and the number of Fourier modes was 64. The domain was split into 65 equal segments and, due to periodicity of the data, only the first 64 pieces of the domain was considered for each of the three dimensions, x, y and z. Then, the frequency components were defined and, since 2π periodic signals are required for the FFT, were scaled by 2π divided by the total length of the spatial domain, 20. They were also shifted using `fftshift()` so that the frequency components would not need to be shifted every time they were plotted. Finally, a 3D grid was created of the shifted frequency components and of the three-dimensional computational domain.

The first goal of this assignment was to find the frequency signature, which was the center frequency, that the submarine produced through averaging the spectrum. In Section 2, we considered the case of a one-dimensional signal. Here, we instead consider a three-dimensional signal, representing each point in space. However, the overall process remains much the same.

First, we define a variable called `ave`, short for average, to be a 3D array of dimension 64 by 64 by 64 (noting that 64 is the previously defined number of Fourier modes) zeros. Next, we create a for loop, looping over values from 1 to the total number of increments of time in the data, 49. (This represents the fact that the data was obtained over a period of 24 hours in half-hour increments; 49 in all.)

Inside this for loop, each column of the submarine data, representing the frequencies of the signal in space at a given time step, was reshaped into a 3D array. This array went through the FFT and afterwards, its values were added to `ave`. In other words, by the end of the for loop, `ave` represented the sum of all the x, y, z values for the signal across all of the time steps.

Then, the variable ave was shifted back into a more usable order of values with the function `fftshift()`. Its absolute value was taken and then, it was divided by the total number of time steps which, again, was 49. Now, `ave` represented the average of all the data points for x, y, z in frequency space. Finally, the result was plotted on a 3D grid whose coordinates were the different shifted frequencies. This algorithm is depicted in the following algorithm graphic, called Algorithm 1.

Algorithm 1: Algorithm for Averaging the Frequency Spectrums

```
Import data from subdata.mat
Define ave, a 3D array of zeros
for j = 1 : timesteps do
    Reshape data at j into 3D array
    Perform fftn() on array
    Add result to ave
end for
fftshift() ave
Divide result's absolute value by timesteps
Plot average signal
```

After the average was obtained and plotted, through visual analysis, I deduced that the center frequency was at about (5,-7,2). These values were then saved into variables, x_0, y_0, z_0 for each dimension and utilized in the selected filter, a Gaussian. As opposed to what was discussed in Section 2, however, the filter had

to work for three dimensions. So, essentially, a separate one-dimensional Gaussian was defined for each dimension and multiplied together. Here is the equation that was used:

$$F(k) = \mathrm{e}^{-\tau(k_x - x_0)^2} \odot \mathrm{e}^{-\tau(k_y - y_0)^2} \odot \mathrm{e}^{-\tau(k_z - z_0)^2} \tag{6}$$

where τ corresponds to the width of the filter, the center of the filter is at (x_0, y_0, z_0), and k_x, k_y, k_z correspond to the frequency grid coordinates for each dimension. The symbol $\odot$ signifies element-wise multiplication. At this point, the value of τ was unknown. It will be further discussed in a following paragraph.

After defining the filter, the algorithm for filtering the data was implemented. In every iteration of a for loop going from 1 to the number of time steps in the data (49) a row of the submarine data was reshaped into a 3D array. This row corresponded to the data for each point in space for a given time step. Then, the matrix was put into frequency space using `fftn()` and shifted to a more usable format using `fftshift()`. This matrix was multiplied by the filter and then put back into the time domain.

At this point, the maximum value of the filtered data in the time domain in absolute value was determined, as well as the index at which it occurred. Using the function `ind2sub()`, the corresponding locations for the x, y, z coordinates were obtained. These were then plugged into the previously defined 3D grid representing the computational domain and plotted. This gave us the coordinates of the submarine's path. The algorithm is depicted in the following graphic, labeled Algorithm 2.

Algorithm 2: Algorithm for Filtering the Frequency Spectrums

```
Import data from subdata.mat
Define filter
for j = 1 : timesteps do
  Reshape data at j into 3D array
  Perform fftn() on array
  fftshift() array
  Multiply filter by array, element-wise
  Perform ifftn() on array
  Locate maximum data point in absolute value
  Determine coordinates corresponding to maximum
  Plot result
end for
```

As mentioned previously, it was necessary to determine τ. In order to do this, the entirety of Algorithm 2 (minus reading in the data) was put inside of another for loop that ran for the length of a vector of potential τ values. A different plot was created for each potential value of τ and titled with the corresponding τ value.

At first, the τ values were drawn from a somewhat wide-spanning vector from 0 to 5 in steps of 0.5. The resulting figures were visually compared to find the least chaotic graph. This corresponded to the "best" value for τ which, according to this test, was somewhere around 0.5. Then, the τ vector was changed to reflect this observation, running from 0 to 0.5 in steps of 0.05. After analyzing all of these resulting graphs, $\tau = 0.4$ seemed to correspond to the smoothest-looking graph. So, τ was defined to be 0.4. The code for this process is included in Appendix B. The code for this entire section is there as well.

4 Computational Results

As mentioned in the previous section, certain algorithms were utilized to find the frequency signature, path, and location of the submarine. After the averaging process discussed in Section 3 and Algorithm 1 was complete, a plot of the averaged frequencies was created. As depicted in Figure 1, the frequency signature can be visually deduced to be at around (5,-7,2). This point in space was then used as the center for the 3D Gaussian filter which helped determine the submarine's trajectory.

After the frequency signature was obtained, a filtering algorithm utilizing these results was executed. This algorithm is discussed in Section 3 and Algorithm 2. The results are depicted in Figure 2, which shows the trajectory of the submarine in both 3D (on the left) and 2D (on the right).

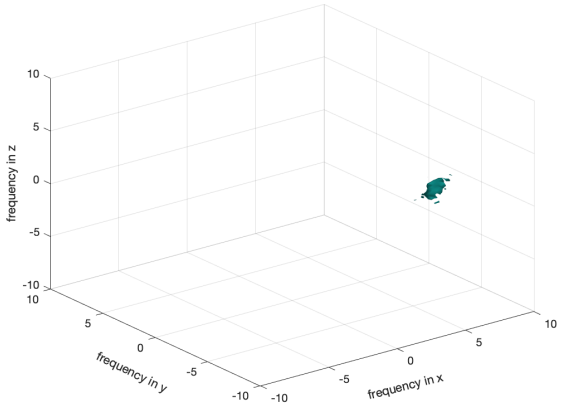

Figure 1: The Submarine's Frequency Signature: Results of Averaging Frequencies

Additionally, the x, y, z coordinates of the path are rounded to two decimal places and shown in Table 1. The final coordinates are rounded and detailed in Table 2. These final coordinates correspond to the submarine's last location. If only the x and y values in these tables are considered (and the z coordinates are ignored), they give the coordinates that would be given to a P-8 Poseidon Aircraft to track the submarine. (Note that the submarine can go underwater, and therefore move in the z direction, and the aircraft cannot. Therefore, the aircraft only needs the x and y coordinates to track the submarine.)

5 Summary and Conclusions

In this assignment, various methods were used to analyze data obtained from a submarine in the Puget Sound. This data was gathered and presented in the form of noisy acoustic data. Through averaging the data in frequency space, the frequency signature emitted by the submarine was discovered. After determining this center frequency, its coordinates were used in a Gaussian filter. The data was then denoised through a filtering algorithm to find the submarine's trajectory and location at the end of data collection. This trajectory was presented in both a 2D and a 3D format. Considering three dimensions gave us the path of the submarine, while considering only two dimensions (i.e. excluding the z coordinates from the analysis) gave the path for an aircraft to find the submarine. These results were presented in both visual formats and in a table.

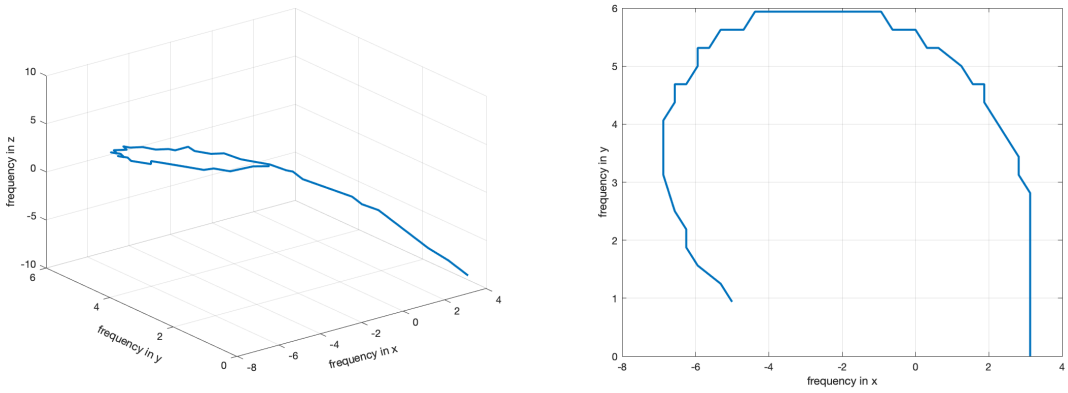

Figure 2: The Submarine's Path: Results of Filtering Data, 3D and 2D View

Time Step	1	2	3	4	5	6	7	8	9	10	11	12
X-Coordinate	3.12	3.12	3.12	3.12	3.12	3.12	3.12	3.12	3.12	2.81	2.81	2.50
Y-Coordinate	0	0.31	0.62	1.25	1.56	1.88	2.19	2.50	2.81	3.12	3.44	3.75
Z-Coordinate	-8.12	-7.81	-7.50	-7.19	-6.88	-6.56	-6.25	-5.94	-5.62	-5.31	-5.00	-4.69

Time Step	13	14	15	16	17	18	19	20	21	22	23	24
X-Coordinate	2.19	1.88	1.88	1.56	1.25	0.62	0.31	0	-0.62	-0.94	-1.25	-1.88
Y-Coordinate	4.06	4.38	4.69	4.69	5.00	5.31	5.31	5.62	5.62	5.94	5.94	5.94
Z-Coordinate	-4.38	-4.06	-3.75	-3.44	-3.12	-2.81	-2.50	-2.19	-1.88	-1.88	-1.25	-1.25

Time Step	25	26	27	28	29	30	31	32	33	34	35	36
X-Coordinate	-2.19	-2.81	-3.12	-3.44	-4.06	-4.38	-4.69	-5.31	-5.62	-5.94	-5.94	-6.25
Y-Coordinate	5.94	5.94	5.94	5.94	5.94	5.94	5.62	5.62	5.31	5.31	5.00	4.69
Z-Coordinate	-0.94	-0.62	-0.31	0	0.31	0.62	0.94	1.25	1.56	1.88	2.19	2.50

Time Step	37	38	39	40	41	42	43	44	45	46	47	48
X-Coordinate	-6.56	-6.56	-6.88	-6.88	-6.88	-6.88	-6.88	-6.56	-6.25	-6.25	-5.94	-5.31
Y-Coordinate	4.69	4.38	4.06	3.75	3.44	3.44	3.12	2.50	2.19	1.88	1.56	1.25
Z-Coordinate	2.81	3.12	3.44	3.75	4.06	4.38	4.69	5.00	5.00	5.62	5.62	6.25

Table 1: Submarine Coordinates for Time Steps 1-48

X-Coordinate	Y-Coordinate	Z-Coordinate
-5.00	0.94	6.56

Table 2: Final Submarine Coordinates (Time Step 49)

Appendix A MATLAB Functions

- `Y = fftn(X)` uses the fast Fourier transform algorithm to return the Fourier transform of an inputted n-dimensional array `X`.
- `Y = fftshift(X)` takes an array `X` in the frequency domain and returns a shifted version of it, such that the element corresponding to the zero-frequency is located in the middle of the array.
- `X = ifftn(Y)` uses the fast Fourier transform algorithm to return the inverse Fourier transform of an inputted n-dimensional array `Y`.
- `[I1,I2,...,In] = ind2sub(sz,ind)` returns `n` arrays with subscripts corresponding to a specified index `ind` for a size `sz` array.
- `fv = isosurface(X,Y,Z,V,isovalue)` uses volume data `V` at designated value `isovalue` to create an isosurface. `X,Y,Z` are `meshgrid` values that represent a grid and the 3D array `V` has the data points corresponding to this grid. This function returns a struct `fv` that contains the data necessary for plotting the isosurface.
- `M = max(A,[],'all')` returns the maximum value contained in `A`.
- `[M,I] = max(A)` returns the maximum value contained in `A` and the index at which the maximum value occurred.
- `[X,Y,Z] = meshgrid(x,y,z)` returns three arrays, `X,Y,Z` that, together, create a 3D grid of size `length(y)`-by-`length(x)`-by-`length(z)`. The grid's coordinates are given by the contents of vectors `x,y,z`.
- `plot3(X,Y,Z,LineSpec)` plots `X,Y,Z` in 3D space corresponding to specifications, such as line width, given by `LineSpec`.

Appendix B MATLAB Code

```
clear all; close all; clc

load subdata.mat % load data file as 262144x49 (space by time) matrix called subdata
timesteps = size(subdata, 2); % number of time increments
L = 10; % spatial domain
n = 64; % Fourier modes
x2 = linspace(-L,L,n+1); % time discretization
x = x2(1:n); y = x; z = x; % take first n points only (periodicity)
k = (2*pi/(2*L))*[0:(n/2 - 1) -n/2:-1]; ks = fftshift(k); % frequency components
[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

% Average the spectrum
ave = zeros(n,n,n);
for j=1:timesteps
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    Unf = fftn(Un);
    ave = ave + Unf;
end
ave = abs(fftshift(ave))/timesteps;

% Plot average signal
M = max(ave,[],'all');
isosurface(Kx,Ky,Kz,abs(ave)/M,0.5)
axis([-10 10 -10 10 -10 10]), grid on, drawnow
xlabel('frequency in x','Rotation',14)
ylabel('frequency in y','Rotation',-26)
zlabel('frequency in z')

% Define filter, centered about (5,-7,2)
coordX = 5; coordY = -7; coordZ = 2; tau = 0.4;
filter = exp(-tau*(Kx - coordX).^2).*exp(-tau*(Ky - coordY).^2).*exp(-tau*(Kz - coordZ).^2);

% Filter data
for j=1:timesteps
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    Unf = fftshift(fftn(Un));
    UnFiltered = filter.*Unf;
    Unt = ifftn(UnFiltered);
    [M,I] = max(abs(Unt(:)));
    [x,y,z] = ind2sub([n,n,n], I);
    pos(j,:) = [X(x,y,z),Y(x,y,z),Z(x,y,z)];
end

% Plot filtering results
figure, plot3(pos(:,1),pos(:,2),pos(:,3),'LineWidth',2), grid on
xlabel('frequency in x','Rotation',14)
ylabel('frequency in y','Rotation',-26)
zlabel('frequency in z')
figure, plot(pos(:,1),pos(:,2),'LineWidth',2), grid on
xlabel('frequency in x')
ylabel('frequency in y')
```

Listing 1: This is the code used for averaging, filtering, and plotting the submarine data.

```
tauVals = 0:.5:5;
% After finding tau=0.5 to be the best fit of this data, tauVals became 0:.05:.5
ind = length(tauVals);

% Loop over values of tau to find best fit
for k=1:ind

    % Define filter
    tau = tauVals(k);
    filter = exp(-tau*(Kx - coordX).^2).*exp(-tau*(Ky - coordY).^2).*exp(-tau*(Kz - coordZ).^2);

    % Filter data
    for j=1:timesteps
        Un(:,:,:)=reshape(subdata(:,j),n,n,n);
        Unf = fftshift(fftn(Un));
        UnFiltered = filter.*Unf;
        Unt = ifftn(fftshift(UnFiltered));
        [M,I] = max(abs(Unt(:)));
        [x,y,z] = ind2sub([n,n,n], I);
        pos(j,:) = [X(x,y,z),Y(x,y,z),Z(x,y,z)];
    end

    % Plot results
    figure
    plot3(pos(:,1),pos(:,2),pos(:,3));
    title(['tau = ', num2str(tau)]);
end
```

Listing 2: This is the code used to determine the value of τ to use in the Gaussian filter function in the main code, listed previously.