AMATH 482 Assignment 1

Megan Freer

January 27, 2021

Abstract

Signals collected in the real world are almost always noisy due to factors such as reflections from geographical objects and atmospheric conditions. As an example, a submarine that emits a particular acoustic frequency can be recorded using a broad spectrum recording of acoustics, but this data is likely noisy and difficult to directly tell the location of the submarine from. In order to solve this issue, we use the Fast Fourier Transform, Gaussian spectral filters, and the Inverse Fast Fourier Transform to filter out noisy data and find locations of objects, such as the location of the submarine over the course of the 24-hour time period for which it was recorded.

1 Introduction and Overview

Almost all signals recorded in the real world are noisy, causing issues for how to analyze them. As a case study, a submarine is moving throughout Puget Sound and is being tracked from noisy acoustic data. There is a new submarine technology that emits an unknown acoustic frequency that needs to be detected. Using a broad spectrum recording of acoustics, data is gathered over a 24-hour period in half-hour increments for a total of 49 time points. The submarine must be located by its coordinate points at each time point in order to find its trajectory.

In order to process this acoustic data, we will be using MATLAB to perform Fast Fourier Transforms (FFT), Inverse Fast Fourier Transforms, and Gaussian spectral filters to de-noise the signal, all of which are important tools in signal processing. By finding the frequency signature (center frequency) generated by the submarine, we were then able to filter around that frequency in each direction (x, y, z). For each time point, the data had a 3D Fast Fourier Transform (FFT) performed on it, the data was then filtered in the frequency domain, and then the filtered data was returned to the time/space domain by performing a 3D Inverse Fast Fourier Transform. This produces the de-noised x, y, and z location of the submarine every 30 minutes over the course of the 24 hour time period to reveal a smoothed, spiral trajectory of the submarine.

2 Theoretical Background

The Fourier Transform is one of the most important mathematical tools in signal processing as a way to break down a signal into different frequencies (meaning going from the time/space domain into the frequency domain). Given that f(x) is a function with $x \in \mathbb{R}$, we define the Fourier transform of f(x) by the following formula, where $\hat{f}(k)$ represents the Fourier transform of f(x):

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

It is important to note that the Fourier transform can also be performed in 3D, as is the case with the submarine coordinates in the x, y, and z dimensions. The Fourier transform decomposes the signal into frequencies in each direction. If we have $\hat{f}(k)$ and would like to find f(x), we can use the Inverse Fourier Transform:

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

This Inverse Fourier Transform goes from the frequency domain to the time/space domain. Knowing Euler's formula (see equation 3 below), we know that e^{ikx} is like $sin(kx)$ and $cos(kx)$, so the value of k gives the frequencies of sine and cosine waves.

$$e^{i\theta} = cos(\theta) + isin(\theta) \tag{3}$$

However, the Fourier Transform is not always useful because it assumes an infinite domain, which is not necessarily true in practical situations. For example, we have the submarine data at 49 time points, not over continuous, infinite time. This brings up the need for the Discrete Fourier Transform (DFT) and its faster version, the Fast Fourier Transform (FFT). We can never actually know about extremely high frequency phenomena that is happening between the points of a discrete signal. As a result, we cannot expect information about a signal for large frequency values and the Discrete Fourier Transform is like a Fourier series but truncated at a maximum frequency as a result of this shortcoming. If we are given an array of N values that are a function sampled at equally spaced points $\{x_0, x_1, x_2, ..., x_{N-1}\}$, the discrete Fourier transform is a sequence of numbers given by:

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi i k n}{N}} \tag{4}$$

In this equation, we only have a finite number of frequencies present: $k = 0, 1, 2..., N-1$. When implementing this transform, there is a possibility of aliasing, meaning that k and $k+N$ are the same in the DFT, and it is possible that a signal may be mistaken for a different signal. Due to this, we instead consider the frequencies given by:

$$k = -N/2, -N/2+1, ..., -1, 0, 1, 2, ..., N/2-1 \tag{5}$$

The Fast Fourier Transform (FFT) is an algorithm to do the Discrete Fourier Transform faster. The DFT can be very slow due to the fact that the complexity is $O(N^2)$, while the FFT is much faster with a complexity of $O(Nlog(N))$. There is a big difference between these values when N is big when lots of data is being analyzed. The main idea behind this new method of FFTs is that if we have a sequence of length N, we can split the DFT into two DFTs of sequences of length N/2 and then continue to repeat this process over again. The FFT is the algorithm that will be used in our MATLAB code for Fourier Transforms.

It is also important to understand Gaussian spectral filters as a method of de-noising a signal. A spectral filter around the frequency of the signal can be multiplied by the signal (in the frequency domain) in order to remove undesired "noisy" frequencies and white noise. A Gaussian function is one example a possible filter and is given by:

$$F(k) = e^{-\tau(k-k_0)^2} \tag{6}$$

In this equation, τ determines the width of the filter and the constant k_0 determines the center of the filter. Filter design, shape, and parametrization are all important factors for signal processing applications.

3 Algorithm Implementation and Development

Algorithms involving Fourier Transforms and Gaussian filters were used to locate the submarine. The code starts by loading in the submarine data, which comes in the form of a 262144x49 (space by time) matrix with noisy acoustic data in half-hour increments over the course of 24 hours. This data can be rearranged to be in a more understandable format, creating a 64x64x64 matrix at each time point.

In order to plot this data, we also defined a 1x64 array of frequencies, as defined in Equation 5, due to aliasing. This array of frequencies is then re-scaled by $\frac{2\pi}{L}$ due to the fact that the FFT assumes 2π periodic signals. This array of frequencies was then put into `fftshift` in order to shift the zero-frequency component to the center of the array. Next the `meshgrid` function was used to return 3D grid coordinates defined by the three 1x64 arrays passed in. This is able to create 64x64x64 matrices for each direction (x, y, z) for both space and frequency.

In order to use this data to find the submarine location, we first must average the spectrum in order to get the frequency signature (center frequency) generated by the submarine. To do this, we must initialize a 64x64x64 matrix that can be used to store the sum of the multidimensional Fourier transform of the data at

each time point. Looping through each of the 49 time points, the 262144x1 subdata array at that time point is reshaped into 64x64x64 data. The multidimensional Fourier transform is taken of this 64x64x64 matrix to transform into the frequency domain, and this value is then added to the variable keeping track of the sum of frequencies. After looping through all of the time points, the average frequency can be found by calculating the absolute value of the sum of frequencies divided by the 49 time points to produce the average frequency matrix with dimensions 64x64x64. Next, we find the maximum frequency component (center frequency) of the average frequency matrix and the corresponding index of this maximum frequency. Since the `max` function used gives a linear index, we then convert the linear index into the index in 64x64x64 space to give indexes in (x, y, z). Next, we find the frequency center in the 3D frequency domain using these coordinate indexes and the previously created 64x64x64 frequency matrices. After taking the absolute values of these frequencies, these are the x, y, and z components of the center frequency.

The next major step to locate the submarine is to create a Gaussian spectral filter and filter the data around the center frequency determined above in order to de-noise the data. Three Gaussian filters are created for the x, y, and z directions, with $\tau = 0.5$ (which was determined through trials with various τ values). Each filter was made as a Gaussian function (see equation 6), with the constant center frequency being subtracted from the frequency values for each of the 3 dimensions. The three filters are multiplied together element-wise to create an overall 64x64x64 Gaussian spectral filter. Next, a 3x49 matrix was initialized to store the (x, y, z) location values of the submarine at each of the 49 time points. In order to fill this in with the correct information, we once again loop through each of the 49 time points. We reshape the data to be a 64x64x64 matrix, perform a multidimensional Fourier transform to transform into the frequency domain, and then element-wise multiply the unfiltered frequency data by the filter to produce the filtered frequency data. Next, we must go back to the spacial domain by performing a multidimensional discrete inverse Fourier transform on the filtered frequency data. Similar to when finding the center frequency, we find the index of the maximum frequency at each time point, convert this linear index to the indices in the 64x64x64 space, and then find the corresponding x, y, and z coordinate locations at these indices. These coordinates can then be accessed and plotted in 3D to display where the submarine is located over time.

4 Computational Results

After performing these calculations, the code was able to produce computational results for the position of the submarine in 3D space at each of the 49 time points over the course of 24 hours. These coordinate values can be seen in Table 1. This table only includes the x and y coordinates, because x and y are coordinates on the surface of the earth, while z is depth. Since the P-8 Poseidon subtracking aircraft that is being used to track the submarine goes in the air rather than in the water, the aircraft only needs to know the x and y coordinates of the submarine to track it. For a visualization of these results in 3D space, see Figure 1. The submarine starts at the lowest Z coordinate and travels upward in a smooth spiral over the 24 hour period.

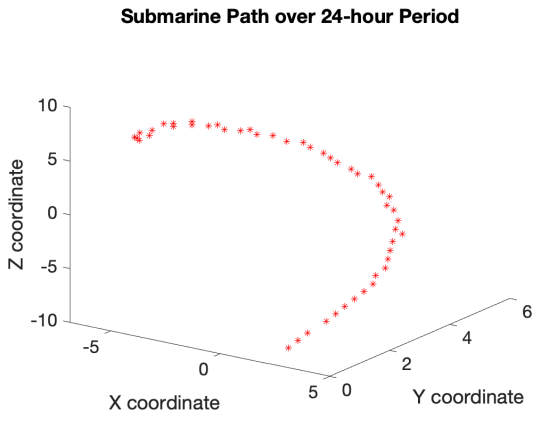

Figure 1: The submarine's calculated position in (x, y, z) over the 24 hour time period (each * represents the location of the submarine at a half-hour interval)

Time (hrs)	x coordinate	y coordinate
0	3.125	0
0.5	3.125	0.3125
1	3.125	0.625
1.5	3.125	1.25
2	3.125	1.5625
2.5	3.125	1.875
3	3.125	2.1875
3.5	3.125	2.5
4	3.125	2.8125
4.5	2.8125	3.125
5	2.8125	3.4375
5.5	2.5	3.75
6	2.1875	4.0625
6.5	1.875	4.375
7	1.875	4.6875
7.5	1.5625	4.6875
8	1.25	5
8.5	0.625	5.3125
9	0.3125	5.3125
9.5	0	5.625
10	-0.3125	5.625
10.5	-0.9375	5.9375
11	-1.25	5.9375
11.5	-1.875	5.9375
12	-2.1875	5.9375
12.5	-2.8125	5.9375
13	-3.125	5.9375
13.5	-3.4375	5.9375
14	-4.0625	5.9375
14.5	-4.375	5.9375
15	-4.6875	5.625
15.5	-5.3125	5.625
16	-5.625	5.3125
16.5	-5.9375	5.3125
17	-5.9375	5
17.5	-6.25	4.6875
18	-6.5625	4.6875
18.5	-6.5625	4.375
19	-6.875	4.0625
19.5	-6.875	4.0625
20	-6.875	3.4375
20.5	-6.875	3.4375
21	-6.875	3.125
21.5	-6.5625	2.5
22	-6.25	2.1875
22.5	-6.25	1.875
23	-5.9375	1.5625
23.5	-5.625	1.25
24	-5	0.9375

Table 1: Calculated X and Y coordinates of the submarine over a 24-hour time period in half-hour increments.

5 Summary and Conclusions

In conclusion, Fast Fourier Transforms, Inverse Fast Fourier Transforms, and Gaussian filters are important tools in signal processing that can be used to analyze and de-noise a signal. These tools were able to find the location in x, y, and z coordinates of a submarine that emits an acoustic frequency over the course of 24 hours. They were able to identify the smoothed, upward spiral path of the submarine using MATLAB with fast computation times. These important signal processing tools can be applied to other situations with 1, 2, or 3 or more dimensions, as demonstrated here, unlocking the possibility for decreased noise and improved object detection in many future applications.

Appendix A MATLAB Functions

The following MATLAB functions are implemented in this code:

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`. This function was used to define an array of x values over a given spatial domain.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array. If `X` is a vector, then fftshift swaps the left and right halves of `X`. This function was used on the array of frequency values before plotting.
- `[X,Y,Z] = meshgrid(x,y,z)` returns 3D grid coordinates based on the coordinates contained in the vectors `x`, `y`, and `z`. X is a matrix where each row is a copy of `x`, `Y` is a matrix where each column is a copy of `y`, and `Z` is a matrix where each layer is a copy of `z`. For example, this function was used to set the 3D coordinate grid in both space and frequency.
- `B = reshape(A, sz1, sz2, sz3)` returns a reshaped version of the matrix A, using the size vector `sz` to define `size(B)`. For example, this function was used to reshape 262144x1 linear data into 64x64x64 space.
- `[max_value, max_index] = max(A, [], 'all', 'linear')` returns the linear index into A that corresponds to the maximum value in the matrix A. This method also returns what this maximum value is.
- `isosurface(X,Y,Z,V,isovalue)` plots isosurface data from the volume data V at the isosurface value specified in isovalue. This means that the isosurface connects points with a specified value, similar to how contour lines connect points of equal elevation. X, Y, and Z represent a Cartesian grid and V contains the corresponding values at these points. Therefore this function is a way to plot a rough estimate of location in space.
- `Y=fftn(X)` returns the multidimensional Fourier transform of an N-D array, meaning transforming the input matrix from the time/space domain to the frequency domain. This is equivalent to computing the 1-D transform along each dimension of X.
- `Y = abs(X)` returns the absolute value of each element in array X.
- `[I1, I2, I3] = ind2sub(sz,ind)` returns the arrays `I1`, `I2`, and `I3`, which contain the equivalent row, column, and layer subscripts corresponding to the linear indices `ind` for a matrix of size `sz`. In the context of this code, this function is used to convert from a linear index to the index in the 64x64x64 space.
- `X = ifftn(Y)` returns the passed in matrix `Y` in the spacial domain by performing the multidimensional discrete inverse Fourier transform on the N-D array. This is equivalent to computing the 1-D inverse transform along each dimension of the passed-in matrix.
- `plot3(X,Y,Z,LineSpec)` plots coordinates in 3D, with the LineSpec parameter providing information about the line style, marker, and color. For example, a LineSpec of `'*r'` would plot red * symbols.

- `x_y_coordinates = table(x_coordinate, y_coordinate)` returns a 49x2 table that contains two columns for the two arrays passed in, in this case with x and y coordinates as the two columns and each row being a timepoint.

Appendix B MATLAB Code

```
% Megan Freer
% AMATH 482
% Assignment 1

%% Code from Assignment Introduction
% Clean workspace
clear all; close all; clc

% Imports the data as the 262144x49 (space by time) matrix called subdata
load subdata.mat

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);

for t=1:49
    Un(:,:,:) = reshape(subdata(:,t),n,n,n);
    M = max(abs(Un),[],'all');
    % isosurface(X,Y,Z,abs(Un)/M,0.7)
    % axis([-20 20 -20 20 -20 20]), grid on
    % drawnow
    % pause(1)
end

%% Part 1: averaging spectrum -> frequency signature (center frequency)

% Intializing sum of frequencies over all time points
sum_freq = zeros(n, n, n);

% Loop over all time points and sum data in frequency domain
for t=1:49
    Un(:, :, :) = reshape(subdata(:,t), n, n, n);
    sum_freq = sum_freq +  fftn(Un); % adds the Fourier transform
end

% Find average from sum of frequncies
average_freq = abs(sum_freq) / 49;

% Find max frequency component (center frequency) and index of max freq
[max_freq, max_index] = max(average_freq, [], 'all', 'linear');
```

```
% Use linear index from above to find 64x64x64 indices
[I1, I2, I3] = ind2sub([n n n], max_index);

% Finding frequency center in 3D frequency domain
Kx_center = Kx(I1, I2, I3);
Ky_center = Ky(I1, I2, I3);
Kz_center = Kz(I1, I2, I3);

% Combining x, y, and z components of center frequency
center_freq = abs(([Kx_center, Ky_center, Kz_center]));

%% Part 2: filter data around center freq -> denoise data -> submarine path

% Create Gaussian spectral filter
tau = 0.5;
filter_x = exp(-tau .* (Kx - Kx_center) .^ 2);
filter_y = exp(-tau .* (Ky - Ky_center) .^ 2);
filter_z = exp(-tau .* (Kz - Kz_center) .^ 2);
filter = filter_x .* filter_y .* filter_z;

locations = zeros(3, 49);
figure()

% Looping through all time points
for t = 1:49
    Un(:, :, :) = reshape(subdata(:,t), n, n, n);

    % Change to frequency domain
    unfiltered_Un_freq = fftn(Un);

    % Apply filter to signal in frequency space
    filtered_Un_freq = filter .* unfiltered_Un_freq;

    % Go back to spacial domain
    % (now ifftn, not ifft like in lecture)
    filtered_Un_space = ifftn(filtered_Un_freq);

    % Find the index of the max frequency at each timepoint
    [max_freq, max_index] = max(abs(filtered_Un_space), [], 'all', 'linear');

    % Use linear index from above to find 64x64x64 indices
    [I1, I2, I3] = ind2sub([n n n], max_index);

    % Save in locations
    locations(1,t) = X(I1, I2, I3);
    locations(2,t) = Y(I1, I2, I3);
    locations(3,t) = Z(I1, I2, I3);

    % Make a plot of the saved locations
    plot3(locations(1,t), locations(2,t), locations(3,t), '*r')
    hold on
end

% Label the figure
```

```
title('Submarine Path over 24-hour Period', 'FontSize', 24)
xlabel('X coordinate', 'FontSize', 20)
ylabel('Y coordinate', 'FontSize', 20)
zlabel('Z coordinate', 'FontSize', 20)
set(gca, 'FontSize', 16)

%% Part 3: give x and y coordinates in a table to follow the submarine

x_coordinate = locations(1,:)';
y_coordinate = locations(2,:)';
x_y_coordinates = table(x_coordinate, y_coordinate);
```