AMATH 482 Homework 1: Acoustic Signal Processing for Submarine Detection

Amelia Nathan

January 27, 2020

Abstract

This paper develops hypothetical foundations of time-frequency signal processing into algorithms to track a submarine and determine its path based on signal data taken over 24 hours. Using Fourier Transforms, averaging, and filtering we are able to successfully determine the submarine's signature frequency and map its pathway while navigating limitations inherent in time-frequency analysis stemming from Heisenberg's Uncertainty Principle.

1 Introduction and Overview

1.1 Introduction to Signal Processing

Time-Frequency signal processing is a versatile area of study with extensive applications including seismic waves, echolocation, vocals, and, as will be the topic in this paper, acoustic wave processing [2]. Time-frequency processing has proven adaptable to situations involving moving signal detection, in part leading to its versatility [2]. However, as signal processing has been used a variety of fields, limitations have also arisen; attaining localized information about both frequency and time is difficult. As will be discussed in part 2, the Heisenberg Uncertainty principle highlights this characteristic issue in signal processing applications [2]. It is therefore an ongoing challenge to best analyze data, especially since time variable data is commonly found in nature and scientific observation [2].

1.2 Problem Overview

This paper highlights the extensive application areas and potential of time-frequency processing by focusing on a specific example: submarine tracking using acoustic data. Presented with 49 data points of a broad spectrum recoding of acoustics taken in 30 minute increments over 24 hours, we are tasked with tracking a moving submarine and determining its location and path. We must also determine the frequency signature that the submarine emits.

2 Theoretical Background

Spectral transforms form a fundamental component of solving problems in applied mathematics; Fourier transforms more specifically, for example, enable representing functions as series of sines and cosines, making data visualization more clear [1]. While there are variations on the Fourier Transform, they are all based on this foundational formula that utilizes integration over an infinite domain [1]:

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

To reverse the effects of the Fourier Transform, we can use the inverse:

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

Following understanding of the Fourier Transform comes a finite representation with the Fourier Series and the application to discrete situations with the Discrete Fourier Transform.
The Fourier Series truncates the Fourier Transform [1]:

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

In Equation 3, a_n, b_n are coefficients that weight each portion of the summation to facilitate an accurate representation of the function in terms of sine and cosine. $\frac{a_0}{2}$ shifts the function vertically. Because of orthogonality properties, we can simplify the integration of the series multiplied by $cos(mx)$ to find formulas to represent coefficients a_n and b_n:

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

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

This process naturally produces a period of 2π for the function; we can therefore generalize the coefficients by situating the Fourier Series so $x \in [-L, L]$. The series is then given by Equation 6 and the coefficients more generally by 7. It is important to note that with the resulting 2π period, it is necessary to rescale the function to fit the desired domain, as will be seen in the implementation section.

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{\frac{in\pi x}{L}} \quad x \in (-L, L] \tag{6}$$

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-i\frac{n\pi x}{L}} dx \tag{7}$$

It is important to note that the coefficients are now complex; Based on Euler's identity,

$$e^{\pm ix} = cos(x) \pm isin(x) \tag{8}$$

we can understand these complex values based on the following findings taken directly from the textbook [1]:

1. c_0 is real and $c_{-n} = c_{n^*}$.
2. When f(x) is even, c_n is real.
3. When f(x) is odd, $c_0 = 0$ and c_n are all strictly imaginary.

Convergence is also important to consider when dealing with Fourier Series; When f(x) is discontinuous, the convergence at a discontinuity is the mean of the values on either side of the discontinuity [1].
The Discrete Fourier Transform applies the transformation to a sinusoidal and cosinusoidal function to instances of finite data at specific points in time.

$$\hat{x}_k = \frac{1}{N} \sum_{i=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}} \tag{9}$$

Seen in Equation 9, the Discrete Fourier Transform is inherently limited by its discrete nature. For n data points we can retrieve n frequencies, without information about frequency behavior between those points [1]. Along with this limitation is the notion of aliasing, which occurs when signals appear the same because high frequencies are lost between sample points and instead indicate that the same frequency occurred when placed under the Discrete Fourier Transform.

Finally, a key theoretical concept underlying the complexity of this problem is the Heisenberg Uncertainty Principle. Founded in quantum mechanics, this principle emphasizes the limitations in attaining exact information about a particle's momentum and position; similarly, it is a characteristic of the Fourier Transform in that we cannot know both the exact time and frequency of a signal [1]. There is a trade off in time-frequency analysis between localizing information in time and achieving accurate frequency information, meaning carefully chosen time windows and frequency spectrum are essential [1].

3 Algorithm Implementation and Development

To solve the problem at hand we use a Gaussian filter, generically represented in 3D by

$$F(k) = e^{(-\tau(kx-kx_0)^2+(ky-ky_0)^2)+(kz-kz_0)^2)}. \tag{10}$$

Tau is the chosen bandwidth of the filter; in this case, we choose 0.2 as an intermediate window considering the above discussion of the uncertainty principle. Before applying the filter to the data, we must average the spectrum as seen in Algorithm 1. Averaging the spectrum eliminates the appearance of white noise as it should have a zero mean [1]. This is based on the fact that, with each realization, white noise and the acoustic signal the submarine emits continue. By averaging the spectrum we are left with the frequency of interest - the acoustic signal from the submarine - because the white noise frequencies will cancel out as they are averaged given their zero mean [1]. Note that the spectrum is averaged in the frequency, not time, domain because white noise affects frequencies over all time.

Before implementing the averaging and filter algorithms, we must of course load the data from `subdata.mat` and initialize the necessary 3D spaces and variables. As seen in Appendix B, we initialize the spatial domain, Fourier modes, and create a 3-dimensional grid space using `meshgrid`. As mentioned earlier in the theoretical section, it is at this point that we must also rescale the frequencies to fit our desired domain.

The Discrete Fourier Transform takes $O(N^2)$ operations, as seen in Equation 9. In our implementation we will therefore use an adapted formula that offers a faster calculation using only O(Nlog(N)) operations: the Fast Fourier Transform, which is built into computing programs like Matlab [1]. The Fast Fourier Transform utilizes a divide and conquer algorithmic approach to continuously split the transform into smaller transforms, making it more efficient for large computing problems [1].

Algorithm 1: Averaging The Frequency Spectrum to Find Centered Signal

```
Import data from subdata.mat
for $j = 1 : 49$ do
    Use fft to transform data into frequency domain
    Sum each frequency to attain the cumulative sum
end for
Use fftshift to reorder data and divide by 49
Find the maximum frequency and corresponding index
Use index to center frequency
```

This first algorithm has helped us answer a key task in the problem as well; the centered frequency indicates the submarine's signature acoustic signal it sends out.

Now that we have averaged the frequency and determined the centered signal, we can build the Gaussian filter and apply it to the data. The Gaussian filter serves our purpose by serving as a multiplier, leading to frequencies outside of our established window to 0. We have chosen τ to be 0.2 for our window in consideration of the trade off discussed earlier; we want to achieve frequency information while retaining as much as we can about time. As seen in Algorithm 2 we loop through 49 times because there are 49 realizations and apply the filter and and determine locations for each one.

Appendix B denotes the variables used to store the results of the second algorithm; the filter process allowed us to develop x, y, and z position vectors to describe the location of the submarine.

Algorithm 2: Applying Filter Function and Determining Location

```
Define $\tau$ and Gaussian filter function
Reorder filter with fftshift
for $j = 1 : 49$ do
  Reshape data to fit 64x64x64 grid
  Apply Fourier Transform to data using fft
  Apply Gaussian filter to signal
  ifft the data back into time-space domain
  Take the maximum of filtered data and corresponding index
  Use index to determine x, y, and z positions of submarine
end for
Use plot3 to visualize submarine path
```

4 Computational Results

The algorithmic approach proved successful in tracking the submarine and determining its pathway. The resulting center frequency, or signature frequency of the submarine, was found to be 176.5852.

To visualize these results I used the plot3 command in matlab as seen in Appendix B. This plotting is based on the locations determined by using our filtering function in Algorithm 2, of which the x and y locations are listed in Table 1.

Time Mark	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 Locations Over 24 Hours

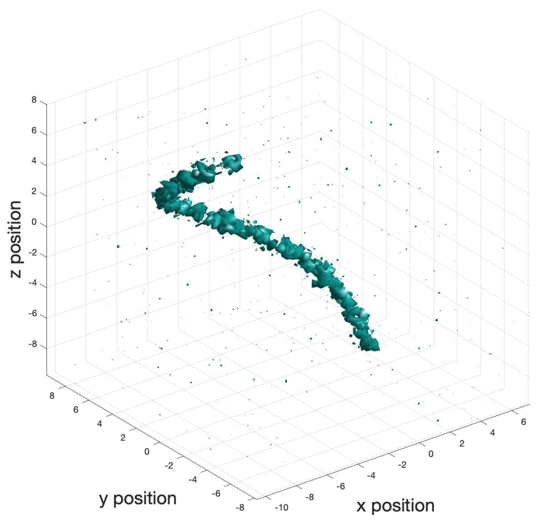

Figure 1: Original Noisy Data

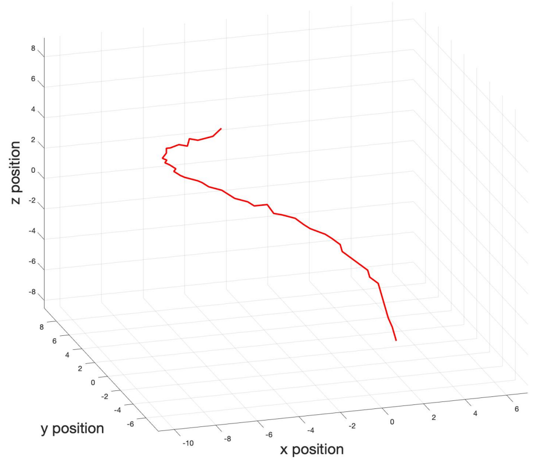

Figure 2: Submarine Path Detected

5 Summary and Conclusions

Through the process of averaging data, applying Fourier Transforms, and using a Gaussian filter function we were able to successfully eliminate noise and pinpoint the submarine's path, as shown in the comparison between Figure 1 and Figure 2. Despite the inherent uncertainty in time-frequency signal processing, we were able to center the frequency and ultimately achieve x, y, and z coordinates for the submarine at each 30 minute increment for the entire 24 hours.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

[2] A Papandreou-Suppappola. *Applications in Time-Frequency Signal Processing (1st ed.)* CRC Press, 2003.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y, Z] = meshgrid(x,y, z)` returns 3-D grid coordinates based on the coordinates contained in the vectors `x`, `y` and `z`. The resulting grid has dimensions length(x) by length(y) by length(z).
- `reshape(X,M,N)` provides an M by N matrix using the columns of X.
- `isosurface(X,Y,Z,V,ISOVALUE)` computes geometric shapes representing the resulting structure from arrays (X,Y,Z) and data V.
- `fftn` is the discrete fourier transform for n dimensions.
- `fftshift` reorders vector based on center of spectrum
- `ifftn` returns inverse discrete Fourier transform

- `ind2sub` returns subscripts correlating to indices
- `max(X)` returns maximum value within X
- `abs` returns absolute value
- `zeros(m, n)` creates a matrix of zeros with m x n dimensions

Appendix B MATLAB Code

```
%% Set up, initialization
% Clean workspace
clear all; close all; clc

load subdata.mat % Imports the data as the 262144x49
% (space by time) matrix called subdata 5

L = 10; % spatial domain
n = 64; % arbitrary value; chosen as power of 2 for efficient execution
x2 = linspace(-L,L,n+1); % Create a vector with 65 equally spaced sections
x = x2(1:n);
y =x;
z = x; % initializing the spatial domain for graph
% Rescale frequency domain to fit our spatial domain
k = (2*pi/(2*L))*[0:(n/2 - 1) -n/2:-1];

ks = fftshift(k); % shift frequencies back to center of spectrum
[X,Y,Z]=meshgrid(x,y,z);

[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

% create initial figure plotting noisy data
for j = 1:49
    Un(:,:,:) = reshape(subdata(:,j),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
    title('Original Submarine Data', 'Fontsize', [26])
    xlabel('x position', 'Fontsize', [20])
    ylabel('y position', 'Fontsize', [20])
    zlabel('z position','Fontsize', [20])
end
%% Algorithm 1 - Averaging

ave = zeros(1,n); % initializes average

for j = 1:49 % Loop through for each realization in the 24 hour period
    Un(:,:,:) = reshape(subdata(:,j),n,n,n);
    Unt = fftn(Un); % to frequency domain
    ave = Unt + ave; % summing frequency realization
end
```

```
ave = abs(fftshift(ave))/49;
[center_signal, index] = max(ave(:)); % find maximum signal
% so we know where to center it (at the index)
[xind, yind, zind] = ind2sub(size(ave), index);
x0 = Kx(xind, yind, zind); % represents centered frequency -
% this is the submarine's signature (x0, y0, z0)
y0 = Ky(xind, yind, zind);
z0 = Kz(xind, yind, zind);

%% Algorithm 2 - Filter Function
xpos = zeros(0,49);
ypos = zeros(0,49);
zpos = zeros(0,49);

tau = 0.2; % assign as window we are keeping to analyze
% Defining filter
filter = exp(-tau*(Kx - x0).^2).*exp(-tau*(Ky - y0).^2).*exp(-tau*(Kz - z0).^2);
filter = fftshift(filter);
% Apply filter at each point in time
for j = 1:49
    Un(:,:,:) = reshape(subdata(:,j),n,n,n);
    Unt = fftn(Un);
    unft = filter.*Unt; % Applying filter to signal
    % inverse fourier transform to get filtered data in space and time
    unf = ifftn(unft);
    [maxintime, index2] = max(unf(:));
    [xind2, yind2, zind2] = ind2sub(size(unf), index2);
    xpos(j) = X(xind2, yind2, zind2);
    ypos(j) = Y(xind2, yind2, zind2);
    zpos(j) = Z(xind2, yind2, zind2);
end
% plot submarine path based on location coordinates
plot3(xpos, ypos, zpos, 'r', 'Linewidth', 2)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
title('Submarine Path', 'Fontsize', [26])
xlabel('x position', 'Fontsize', [20])
ylabel('y position', 'Fontsize', [20])
zlabel('z position','Fontsize', [20])

xycoordinates = [xpos; ypos]; % for table generation
```