AMATH 482 - Assignment 1

Clarisa Leu-Rodriguez

Winter 2021 - January 27th, 2021

Abstract

In this paper, we explore a dataset containing acoustic data from the Puget Sound taken over a 24-hour time period in half-hour increments. Our goal is to determine the location and path of a new submarine which emits an unknown acoustic frequency to send our P-8 Poseidon subtracking aircraft to follow the submarine. This is done through the use of the discrete Fourier transform to convert the given data to the frequency domain, averaging the transformed frequency data to determine the frequency signature of the submarine, filtering the data with a Gaussian function around the frequency signature to amplify the submarines signal and diminish any noise, and mapping the submarines path and location with the transformed, denoised data for the aircraft to the follow.

Section I. Introduction and Overview

We are given noisy acoustic data taken from the Puget Sound where a submarine is suspected to be located. We know the submarine is using a new technology which emits an unknown frequency that we haven't yet detected before. The noisy data that was obtained was taken over a 24-hour period in half-hour increments (i.e. - the data represents 49 points in time or "realizations"). In Figure 1 below, we plot the isosurface of the original data.

As the acoustic signature of the submarine is unknown - we assume that the frequencies in the data are random. In order to locate the submarine, we will first transform the data from the spatial domain to the frequency domain. This will allow us to average the spectrum which should diminish any random frequencies in the data, and allow us to determine the frequency signature of the new submarine technology. Next, we can apply a Gaussian filter (centered around the submarine's frequency signature) to our data in the frequency-domain, which will help us further isolate the submarine's signal within the noisy data. After applying a Gaussian filter to our frequency-domain data - we can then convert our data back to the spatial domain to determine the location and path of the submarine.

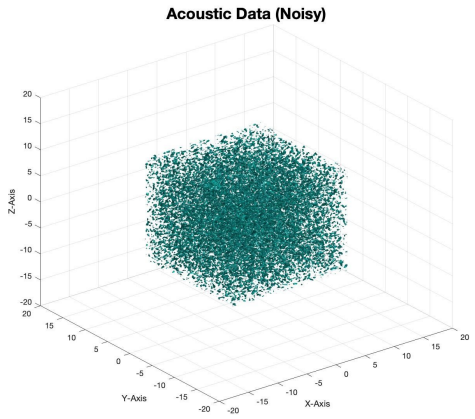

Figure 1: Isosurface visualization of the given acoustic data with noise.

Section II. Theoretical Background

To successfully locate the submarine in the noisy acoustic data and send our P-8 Poseidon subtracking aircraft - we use the Fourier transform (computationally, we use the fast Fourier transform), Gaussian filtering, and signal averaging. In this section, I will discuss the mathematics and theory behind our solution.

Section II.I. The Fourier Transform and Inverse Fourier Transform

In order to analyze the acoustic data in the frequency domain, we use the Fourier transform to decompose the acoustic signals of our data into its frequency components (i.e. sine and cosine waves). Formally, given a periodic function $f(x)$ with $x \in R$, the Fourier transform of $f(x)$, written $\widehat{f}(k)$, is defined by the formula:

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

Equation 1: The Fourier transform of $f(x)$.

Additionally, we use the inverse Fourier transform to transform from the frequency domain back to the spatial domain, where given $\widehat{f}(k)$ we can recover $f(x)$:

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

Equation 2: The inverse Fourier transform of $\widehat{f}(k)$.

The Fourier Transform is widely used in signal processing as we can move back and forth easily between the frequency and time or spatial domain. This is powerful as it allows us to analyze signal data in a different domain, making difficult problems to solve in one domain much easier to analyze and solve in another domain.

Section II.II. The Fast Fourier Transform and Inverse Fast Fourier Transform

Computationally, we use the fast Fourier transform (FFT) algorithm which computes the discrete Fourier transform (DFT) of a function $f(x)$ over a finite interval. More formally, given a function $f(x)$ with $x \in [-L, L]$ being defined on a finite spatial domain where the last point is the same as the first (i.e. periodic) - the DFT is defined as the following:

$$\widehat{x_k} = \sum_{n=0}^{N-1} x_n \cdot e^{\frac{-pi \cdot 2ikn}{N}}$$

Equation 3: The discrete Fourier transform of a discrete function of $f(x)$ - defined over N points.

The inverse of the DFT is also defined as the following:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} \widehat{x_k} \cdot e^{\frac{pi \cdot 2ikn}{N}}$$

Equation 4: The inverse discrete Fourier transform of a discrete function $\widehat{f(k)}$ in the frequency domain.

The FFT improves the runtime from $O(n^2)$ to $O(nlogn)$ when compared to the normal Fourier transform. This is because the FFT Algorithm breaks up the transform of the discrete function $f(x)$ defined on $x \in [-L, L]$ in half continuously and exploits the symmetry of the discrete Fourier transform which leads to fewer computations overall. This increases the speed of considerably, especially for large n. In addition - in most real world applications we are not given a function to work with but instead a set of points to investigate and our data can be quite large in size - which makes the use of the FFT ideal in practical settings.

Section II.III. Signal Averaging and Gaussian Filtering

To remove noise from our data, we filter the data around a desired frequency using a Gaussian filter. To locate the frequency signature of our submarine (i.e. the frequency we are interested in filtering around) we first average the frequency data over each timestep in our transformed data. Since white noise is expected to contribute an average of zero to each frequency by nature, averaging our frequency data will leave us with the frequency of only the submarine's signal in our data.

After locating the frequency signature of the submarine through signal averaging, we use a Gaussian filter to filter around this frequency. More formally - the general Gaussian filter for spectral filtering is defined as the following, where k_0 is the center of the filter and τ is the width of the filter.

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

Equation 5: The equation for the general Gaussian filter.

A Gaussian filter will be centered around k_0 with a normalized value of 1 and will decay to 0 when moving away from the center - i.e. we will amplify frequencies close to the center and decay the ones which are further away from the center. The width of the filter is determined by τ. Gaussian filters are often used to filter signals in the frequency domain due to its smooth nature and also the fact that the Fourier transform of a Gaussian is also a Gaussian - letting us transform back easily.

Section III. Algorithm Implementation and Development

To start this problem, we first import the 64^3-by-49 noisy data into MATLAB and reshape the data into a 64-by-64-by-64 3-Dimensional Matrix - where the first, second, and third coordinate define the location of the acoustic signal (x, y, z) and the value is the measured acoustic signal from our sonar reading. This makes the data easier to reason about and visualize in a 3D Cartesian Coordinate System.

Next, we define a 3D grid in normal space with the X, Y, Z coordinates of our spatial-domain - in addition we define another 3D grid with the corresponding Kx, Ky, Kz coordinate of our frequency-domain. Note that we rescale the wave numbers here by $2 \cdot pi/L$ as the FFT assumes $2 \cdot pi$ periodic frequencies.

After importing and manipulating the data into an easier to understand format, we perform a FFT on the 3-Dimensional data (which gives us the frequencies in 3-Dimensions) and begin to average our data in the frequency domain to get rid of any white noise in our data and locate the frequency signature of the submarine. After averaging our 3-Dimensional frequency data, we normalize the data with the largest frequency in absolute value in our transformed data. This gives us the frequency signature of the submarine - which we use via our Gaussian filter to filter our noisy data around.

After applying our Gaussian filter around the frequency signature of the submarine - we are then able to convert our denoised frequency data back to the time domain using the inverse fast Fourier transform, which gives us the filtered data in the spatial domain with the submarines frequencies amplified and noise diminished. Picking an appropriate width for our filter - we choose 0.5 as it gives us the most smooth looking data. To get the positional coordinates of the submarine for each time slice, we find the maximum value in the filtered time-domain data and collect the (x, y, z) coordinate for each time slice. After doing this for each time step, we then have the positional coordinates for the submarine's path over the 49 time slices for our P-8 Poseidon subtracking aircraft.

Section IV. Computational Results

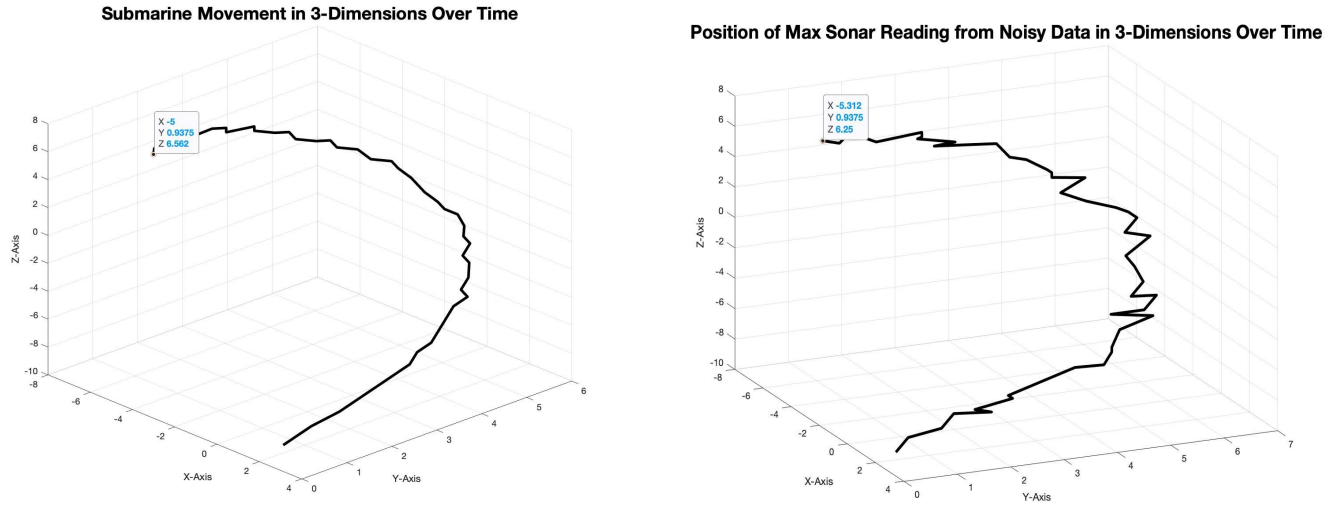

Figure 2: The image on the left (Figure 2a) shows the submarine's path in 3-Dimensional Space over the 49 time slices using our filtered data. The ending position of the submarine in Figure 2a is (-5, 0.9375, 6.5625). The image on the right (Figure 2b) shows the position of the max sonar reading in the noisy, unfiltered acoustic data over the 49 time slices. The ending position of the path of the submarine in Figure 2b is (-5.312, 0.9375, 6.25).

(3.125, 0)	(3.125, 0.3125)	(3.125, 0.625)	(3.125, 1.25)	(3.125, 1.5625)	(3.125, 1.875)	(3.125, 2.1875)
(3.125, 2.5)	(3.125, 2.8125)	(2.8125, 3.125)	(2.8125, 3.4375)	(2.5, 3.75)	(2.1875, 4.0625)	(1.875, 4.375)
(1.875, 4.6875)	(1.5625, 4.6875)	(1.25, 5)	(0.625, 5.3125)	(0.3125, 5.3125)	(0, 5.625)	(-0.3125, 5.625)
(-0.9375, 5.9375)	(-1.25, 5.9375)	(-1.875, 5.9375)	(-2.1875, 5.9375)	(-2.8125, 5.9375)	(-3.125, 5.9375)	(-3.4375, 5.9375)
(-4.0625, 5.9375)	(-4.375, 5.9375)	(-4.6875, 5.625)	(-5.3125, 5.625)	(-5.625, 5.3125)	(-5.9375, 5.3125)	(-5.9375, 5)
(-6.25, 4.6875)	(-6.5625, 4.6875)	(-6.5625, 4.375)	(-6.875, 4.0625)	(-6.875, 4.0625)	(-6.875, 3.4375)	(-6.875, 3.4375)
(-6.875, 3.125)	(-6.5625, 2.5)	(-6.25, 2.1875)	(-6.25, 1.875)	(-5.9375, 1.5625)	(-5.625, 1.25)	(-5, 0.9375)

Table 1: The positional coordinates (x, y) of the submarine's path in the spatial domain from our transformed data. The upper left hand corner corresponds to the first time slice, while the bottom left hand corner corresponds to the 49th time slice. The time slices increase from left-to-right - e.g. the start of the second row corresponds to the 8th time slice.

In Figure 2, we use the `plot3` function in MATLAB to plot the submarine's position over time using our filtered data which is centered around the frequency signature of our submarine (Figure 2a) and determine the ending position of our submarine to be (-5, 0.9375, 6.5625). On another note, Figure 2b is interesting to compare our final results to, as we find that in our noisy data if, for each time slice, we find the positional coordinates of the acoustic signal with the maximum value - we also get a somewhat similar path with similar ending positions. The frequency signature of the submarine was also determined to be (5.340708, -6.911504, 2.199115) after averaging the spectrum and normalizing by the peak of the spectrum. Table 1 shows all of the positional (x, y) coordinates for each time slice of submarine from our denoised, transformed data for our P-8 Poseidon subtracking aircraft.

Section V. Summary and Conclusions

Through the use of the FFT, inverse FFT, averaging the spectrum of our signal data in the frequency domain, and Gaussian filtering to find a submarine in noisy acoustic data taken in the Puget Sound - we determine the frequency signature of the new submarine technology to be (5.340708, -6.911504, 2.199115) and the ending position of the submarine to be (-5, 0.9375, 6.5625). We will send our P-8 Poseidon subtracking aircraft along the path defined by the (x, y) coordinates listed in Table 1 to follow the submarine.

In regards to this problem, we end off by noting that the data set given was simplified in many ways - one being that the submarine was only emitting one type frequency. The problem would be more complicated if it was emitting several different types of frequencies. In addition, the data could be more complex if there was another frequency being emitted from a different source - instead of the data just containing sonar reading from only the submarine with additional white noise.

Section VI. References

- https://en.wikipedia.org/wiki/Discrete_Fourier_transform - I used this source to better understand the discrete Fourier transform. Equation 3 and Equation 4 are from this website.
- In addition, the course notes provided on the course website as well as lecture material was used.

Appendix A. MATLAB Functions Used and Brief Implementation Explanation

This appendix contains a list of the MATLAB functions used and additional information regarding how it was used to solve our submarine problem. Additional information can also be found in Section II. Theoretical Background and also Section III. Algorithm Implementation and Development. Note, descriptions of functions were sourced from the MATLAB documentation which can be found here online: https://www.mathworks.com/help/matlab/.

- **linspace**(X1, X2, N) generates a vector of N equally spaced points from X1 to X2. We used this function to create a linearly spaced vector in the spatial domain (where the last point is the same as the first), which we then used for the meshgrid to generate a 3-D grid coordinate system defined by this vector .
- [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 size length(y)-by-length(x)-by-length(z). As stated above, we used the meshgrid function to create a grid with the desired dimensions in the spatial and frequency domain.
- **load**(FILENAME) loads data from a MAT-file into workspace. We used this command to load our acoustic data into MATLAB.
- **fftshift**(X) shifts the zero-frequency component to the center of the spectrum. If X is a vector, it swaps the left and right halves of X. For matrices, the function fftshift swaps the first and third quadrants and the second and fourth quadrants. For N-D arrays, fftshift swaps "half-spaces" of X along each dimension. We used fftshift to better visualize the Fourier transform with the zero-frequency component in the middle of the spectrum.
- **zeros**(M, N, P, ...) generates an M-by-N-by-P-by-... array of zeros. We used this to create a zero-initialized array to be used to store our transformed data. We use the zeros function in MATLAB to preallocate memory before computation - which saves the computer time as it doesn't need to grab more buckets during computation as the space to store the result has already been preallocated.
- **reshape**(X, M, N, P, ...) returns an N-D array with the same elements as X but reshaped to have the size M-by-N-by-P-by-.... We used reshape to reshape our original data from a 64^3-by-49 vector to a 64-by-64-by-64 matrix where the value contained in the matrix represents the corresponding acoustic reading, etc., and the corresponding position in the matrix represents where that reading was taken in 3-D space.
- [M, I] = **max**(X) returns the largest element in the vector X (M) and also the index I corresponding to its position in X. We used this to find the largest value in our data for each time slice for both the filtered and unfiltered data for plotting. Note that we can convert a matrix A into a vector by doing A(:) which you will see in our code. With the index I, we were able to determine the corresponding subscripts equivalent to I in an N-D array of size(X) using the ind2sub function (see below).

- **abs**(X) is the absolute value of the elements of X. When X is complex, it computes the complex modulus (magnitude) of the elements of X. We used this in our code when we wanted to find the maximum value in magnitude within our data either for our noisy or denoised data - and in addition when averaging the spectrum.
- [I1, I2, I3, …, In] = **ind2sub**(SIZ, IND) returns N subscript arrays I1, I2, …, In containing the equivalent N-D array subscripts equivalent to IND for an array of size SIZ. As stated above, this was used to find the equivalent subscripts in 3-Dimensional space of the maximum value in our data for each time space - allowing us to locate the coordinates of this maximum value within our defined spatial and frequency domain.
- **fftn**(X) returns the N-dimensional discrete Fourier transform of the N-D array X. We used this to find the 3-Dimensional discrete Fourier transform of our data - going from the spatial domain to the frequency domain.
- **ifftn**(F) returns the N-dimensional inverse discrete Fourier transform of the N-D array F. We used this to find the 3-Dimensional inverse discrete Fourier transform of our frequency data - going from the frequency domain back to the spatial domain after applying our filter to our frequency data and amplifying the submarine's frequency.
- **plot3**(x, y, z), where x, y, and z are three vectors of the same length, plots a line in 3-space through the points whose coordinates are the elements of x, y, and z. We used this function to plot the path of the submarine over 49 data points which we obtained from our denoised data. In addition, we used this to plot the path in our noisy data containing the coordinate of the max acoustic sonar reading for each time slice in order to compare that to our denoised data.
- **isosurface**(X, Y, Z, V, ISOVALUE) computes isosurface geometry for data V at isosurface value ISOVALUE. Arrays (X, Y, Z) specify the points at which the data V is given. We used this to visualize our data in the spatial and frequency domains.

Appendix B. MATLAB Codes

This appendix contains the MATLAB code used in this paper. Please refer to Appendix A. MATLAB Functions Used and Brief Implementation Explanation for more information regarding how each the MATLAB functions used in our code below works.

```
%{
    Assignment #1 - A Submarine Problem
    AMATH482 - Computational Methods For Data Science -January 27th, 2021
    Taught by Professor Jason J. Bramburger (Winter 2021)
    Written By: Clarisa Leu-Rodriguez - email: cleu@uw.edu
%}

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
realizations = 49;  % The number of "time slices" in our data.

% We assume the domain is periodic so the last point is the same as the
% first.
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);  % X, Y, Z - spatial domain
[Kx, Ky, Kz] = meshgrid(ks, ks, ks);  % Kx, Ky, Kz - frequency domain

x_pos_noisy = zeros(1, realizations);
y_pos_noisy = zeros(1, realizations); z_pos_noisy = zeros(1, realizations);
unt_ave = zeros(n, n, n);
for j = 1:realizations
    un(:, :, :) = reshape(subdata(:, j), n, n, n);
    [max_val_dat_noise, k] = max(abs(un(:)));
    [max_val_x, max_val_y, max_val_z] = ind2sub(size(un), k);
    x_pos_noisy(j) = X(max_val_x, max_val_y, max_val_z);
    y_pos_noisy(j) = Y(max_val_x, max_val_y, max_val_z);
    z_pos_noisy(j) = Z(max_val_x, max_val_y, max_val_z);
    unt_ave = unt_ave + fftshift(fftn(un));
end
unt_ave = abs(unt_ave) ./ realizations;  % Average the spectrum

% Plot the position of the max sonar reading for each time slice in
% the noisy data to compare to denoised data later.
figure(1)
plot3(x_pos_noisy, y_pos_noisy, z_pos_noisy, 'k-', 'LineWidth', 3);
set(gcf, 'position', [400, 300, 600, 500]);
```

```
title('Position of Max Sonar Reading from Noisy Data in 3-Dimensions Over
Time', ...
    'Fontsize', 20);
xlabel('X-Axis');  ylabel('Y-Axis');  zlabel('Z-Axis');
grid on, hold on
% Plot the last coordinate
plot3(x_pos_noisy(realizations), y_pos_noisy(realizations), ...
    z_pos_noisy(realizations), 'ro');
fprintf('Ending Position From Noisy Data is: (%f, %f, %f). \n', ...
    x_pos_noisy(realizations), y_pos_noisy(realizations), ...
    z_pos_noisy(realizations));

% Look at the isosurface of the noisy data.
figure(2)
set(gcf, 'position', [400, 300, 600, 500]);
isosurface(X, Y, Z, abs(un), 0.4, 'r'); grid on;
title('Acoustic Data (Noisy)','Fontsize', 20); xlabel('X-Axis');
ylabel('Y-Axis'); zlabel('Z-Axis');
axis([-20 20 -20 20 -20 20]);

% Find the peak in our data within the frequency domain by normalizing.
max_val = max(unt_ave(:));
unt_ave = unt_ave ./ max_val;

% Find frequency signature of submarine.
i = find(unt_ave == 1);
Kx0 = Kx(i);  Ky0 = Ky(i);  Kz0 = Kz(i);
fprintf('Frequency Signature of Submarine is: (%f, %f, %f). \n', Kx0, ...
            Ky0, Kz0);

% Remove noise from data and center around frequency
tau = 0.5;  % Pick a bandwidth which makes data look "smoothest"
filter = exp(-tau.*((Kx-Kx0).^2 + (Ky-Ky0).^2 + (Kz-Kz0).^2));

x_pos = zeros(1, realizations); y_pos = zeros(1, realizations);
z_pos = zeros(1, realizations);
for j = 1:realizations
    un_filter(:, :, :) = fftshift(fftn(reshape(subdata(:, j), n, n, n)));
    % Apply filter
    unt_filter(:, :, :) = un_filter.*filter;
    sub(:, :, :) = ifftn(unt_filter);
    % Find the max value for each time slice and get positional
    % coordinates to graph.
    [max_val_dat, k] = max(abs(sub(:)));
    [max_val_x, max_val_y, max_val_z] = ind2sub(size(sub), k);
    x_pos(j) = X(max_val_x, max_val_y, max_val_z);
    y_pos(j) = Y(max_val_x, max_val_y, max_val_z);
    z_pos(j) = Z(max_val_x, max_val_y, max_val_z);
end
```

```
% Plot the submarines position over time.
figure(3)
plot3(x_pos, y_pos, z_pos, 'k-', 'LineWidth', 3);
set(gcf, 'position', [400, 300, 600, 500]);
title('Submarine Movement in 3-Dimensions Over Time', 'Fontsize', 20);
xlabel('X-Axis');  ylabel('Y-Axis');  zlabel('Z-Axis');
grid on, hold on
% Plot the ending position of the submarine.
plot3(x_pos(realizations), y_pos(realizations), z_pos(realizations), 'ro');
fprintf('Ending Position of Submarine is: (%f, %f, %f). \n', ...
    x_pos(realizations), y_pos(realizations), z_pos(realizations));


% Output the submarines positional coordinates.
for j = 1:realizations
    fprintf('Time Slice: %f, Position: (%f, %f, %f)\n', j, x_pos(j), ...
        y_pos(j), z_pos(j));
end
```