Hunting for a submarine

Yumin Yin

January 27, 2021

Abstract

We are hunting for a submarine using acoustic data. Unfortunately, the collected data is mixed with noise, and the acoustic frequency emitted by the submarine is unknown. Through averaging of the spectrum, we first retrieve the center frequency. Next, to denoise the data, we filter the data around the center frequency. Lastly, we determine the path of the submarine and determine its final location to send our P-8 Poseidon subtracking aircraft.

1 Introduction and Overview

We are hunting for a submarine in the Puget Sound using noisy acoustic data. It is a new submarine technology that emits an unknown acoustic frequency that we need to detect. We have successfully obtained data over a 24-hour period in half-hour increments by using a broad spectrum recording of acoustics. Our next goal is to locate the submarine and find its trajectory using the acoustic signature.

To begin, we first apply the fast Fourier transform to convert the data into the frequency domain. After that, to take the advantage of the key that the noise is white, i.e., it has zero mean, we average over realizations (in the frequency domain) to cancel out the noise in order to figure out the center frequency. Lastly, the trajectory of the submarine, including its final location, can be determined by filtering the data around the center frequency.

2 Theoretical Background

2.1 Fourier Transform

Given a function $f(x)$ with $x \in \mathbb{R}$, we define the *Fourier transform* of $f(x)$, written as $\hat{f}(k)$, by the formula(1).

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

Furthermore, given $\hat{f}(k)$ and to recover $f(x)$, we can use the *inverse Fourier transform* (2):

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

Based on *Euler's formula* (3):

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

where $i = \sqrt{-1}$ is the imaginary constant, e^{ikx} is like $sin(kx)$ and $cos(kx)$, which the value of k gives the frequencies of sine and cosine waves. Therefore, the Fourier transform takes a function of space (or time), x, and converts it to a function of frequencies, k. However, the main drawback of the Fourier transform is it assumes an infinite domain, so it is not necessarily what we want for the practical situations. This leads us to consider something that works for finite domains: Fourier series.

2.2 Fourier Series

Suppose we are given a function $f(x)$ for a limited range of x values. For simplicity, we will take $x \in (-\pi, \pi]$. The first thing to note is that not all frequencies 'fit' into this interval. That is, we will only consider sin(kx) and cos(kx) with integer k since these are the only frequencies that lead to functions that completely repeat over the interval. This leaves infinitely many frequencies, given by k = 1,2,3,4,..., and we write $f(x)$ as the infinite sum of these sines and cosines (4):

$$f(x) = \frac{a_0}{2} + \sum_{i=1}^{\infty} (a_k \cos kx + b_k \sin kx) \quad x \in (-\pi, \pi]. \tag{4}$$

The Fourier coefficient take the forms of the equations (5), (6), and (7):

$$a_k = \frac{1}{L} \int_{-L}^{L} f(x) \cos(\frac{\pi kx}{L}) dx, \tag{5}$$

$$b_k = \frac{1}{L} \int_{-L}^{L} f(x) \sin(\frac{\pi kx}{L}) dx, \tag{6}$$

$$a_0 = \frac{1}{2L} \int_{-L}^{L} f(x) dx, \tag{7}$$

2.3 Discrete Fourier Transform(DFT)

Suppose we are given a sequence (or vector) 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 (8)

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi i k n}{N}}. \tag{8}$$

Here we have only finitely-many frequencies present: the ones given by $k = 0, 1, 2, ..., N-1$. However, there is something tricky here that is important when we get to the MATLAB implementation. Since we are only sampling a signal at discrete values, it is possible that we may mistake it for another different signal. This is referred to as `aliasing` and comes from the fact that $-k$ and $k+N$ are the same in the DFT. To make things more confusing, MATLAB gives the DFT values in this order: $\{\hat{x}_0, \hat{x}_1, ..., \hat{x}_{N/2-1}, \hat{x}_{-N/2}, \hat{x}_{-N/2+1}, ..., \hat{x}_{-1}\}$.

2.4 Faster Fourier transform(FFT)

The FFT is an algorithm that does the DFT faster. The DFT calculates each number takes N operations, and its complexity is $\mathcal{O}(N^2)$, as we need to compute N numbers, where as the FFT's is $\mathcal{O}(Nlog(N))$. The basic idea of this FFT algorithm is that if we have a sequence of length N, we can split the DFT into two DFTs of sequence of length $N/2$ and repeat this process over and over. Luckily, MATLAB has a built-in function to perform the FFT, which we mainly use for our task.

2.5 Averaging

The good news is that the noise is white, i.e., it has zero mean. Therefore, if we average over many realizations, the noise from each realization will cancel out and we can get something close to the true signal.

2.6 Filtering

We can use a spectral filter around the center frequency, which can be determined by the technique of averaging, to remove undesired frequencies. Mathematically, a filter is just a function that we multiply the signal by (in the frequency domain). There are many options for filters. Let's use a simple one, the Gaussian function (9):

$$F(k) = e^{-\tau(k-k_0)^2}, \tag{9}$$

where τ determines the width of the filter, and k_0 is the center of the filter.

3 Algorithm Implementation and Development

3.1 Preparing the Data (Lines 1-11)

The frequency domain, k, is re-scaled by $2\pi/L$ since the MATLAB built-in fuction FFT assumes 2π periodic signals. A 3-D grid of X,Y,and Z coordinates is created to represent the spatial domain defined by the vector of x, y, and z. Similarly, a 3-D grid of Kx, Ky, and Kz coordinates is created to represent the frequency domain.

3.2 Averaging of the Spectrum (Lines 13-26)

At each time measurement, i.e., each column, the given data is reshaped into a $64 \times 64 \times 64$ array and then transformed to the corresponding frequency domain by applied fftn function, which computes the Fourier transform in higher dimensions. It is important to apply the fftshift function to rearrange the data by shifting the zero-frequency component to the center. Since there are complex values given in the data, we use the abs function to compute complex magnitude. We are now ready to cancel out the noise by averaging over realizations in the frequency domain. As there is no more noise, we can locate where the frequency signature is in the frequency domain by determining the maximum frequency.

3.3 Filtering the Data (lines 29-40)

We first define our filter as the Gaussian function around the center frequency in 3-D with $\tau = 0.2$. At each time measurement, we reshape the given data into a $64 \times 64 \times 64$ array and apply the 3D Fourier transform to convert the data to the corresponding frequency domain. We then multiply the data with the Gaussian function, which is previously defined. This is how we filter the data. After that, we apply the 3D inverse Fourier transform to transform our data back to the spatial domain. It is worth mentioning that we should also rearrange the data by shifting the zero-frequency component to the center and compute complex magnitude each time when it is needed. Lastly, we can track the path of the submarine by looking for where the maximum frequency, which is the signal emitted by the submarine, is in the spatial domain at each time measurement.

3.4 Plotting (lines 43-50)

we plot the trajectory of the submarine over time and determine its final location.

4 Computational Results

4.1 The Frequency Signature

The frequency signature generated by the submarine is detected at [5.3407, -6.9115, 2.1991] in the frequency domain.

4.2 The Path of the Submarine

The figure 1 indicates the location of the submarine at each time measurement and depicts the path of it over time.

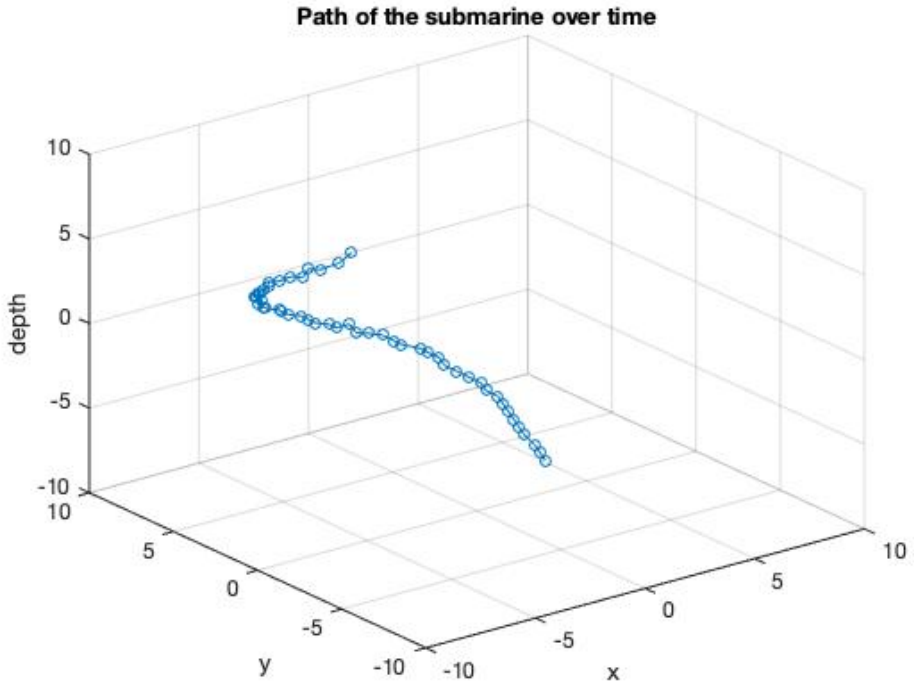

Figure 1: the trajectory of the submarine over time

x-coordinate	y-coordinate
-5.0000	0.9375

Table 1: x and y coordinates of the final location of the submarine

4.3 The Final Location

The table 1 shows the x and y coordinates of the final location of the submarine, and this is where we should send our P-8 Poseidon subtracking aircraft.

5 Summary and Conclusions

We have succeeded in locating the submarine and finding its trajectory using the acoustic signature. We used the Fourier transform and inverse Fourier transform to convert the data in the spatial domain to the frequency domain back and forth in favor of us. Through averaging of the spectrum, we determined the frequency signature (center frequency) generated by the submarine in the frequency domain. After that, we filtered the data around the center data and locate the maximum frequency, i.e., the submarine in the spatial domain at each time measurement in order to figure out the path of the submarine over time.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y] = meshgrid(x,y)` returns 2-D grid coordinates based on the coordinates contained in the vectors `x` and `y`. X is a matrix where each row is a copy of `x`, and `Y` is a matrix where each column is a copy of `y`. The grid represented by the coordinates `X` and `Y` has `length(y)` rows and `length(x)` columns.
- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `Y = fftn(X)` returns the multidimensional Fourier transform of an `N-D` array using a fast Fourier transform algorithm.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `M = max(A,[],'all')` finds the maximum over all elements of `A`.
- `Y = abs(X)` returns the absolute value of each element in array X. If `X` is complex, `abs(X)` returns the complex magnitude.
- `find (X == Y)` returns the indices where `X = Y`.
- `[I1,I2,...,In] = ind2sub(sz,ind)` returns `n` arrays `I1,I2,...,In` containing the equivalent multidimensional subscripts corresponding to the linear indices `ind` for a multidimensional array of size `sz`. Here `sz` is a vector with `n` elements that specifies the size of each array dimension.
- `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 coordinates in `3-D` space.

Appendix B MATLAB Code

```
clear all; close all; clc

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); % returns 3-D grid coordinates defined by the vectors x, y, and z
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

% Average of the spectrum
ave = zeros(n,n,n);
for i = 1:49
    ave = ave + fftn(reshape(subdata(:,i),n,n,n)); % reshape and apply the Fourier transform
end
ave = abs(fftshift(ave))/49; % shift the Fourier transform data and take an average over realizations

% Determine the center frequency and its location in the frequency domain
maxAve = max(abs(ave),[],'all');
[x_index,y_index,z_index] = ind2sub(size(ave), find(abs(ave) == maxAve));
maxFrequencyAt = [Kx(x_index,y_index,z_index),Ky(x_index,y_index,z_index),Kz(x_index,y_index,z_index)];
x0 = Kx(x_index,y_index,z_index)
y0 = Ky(x_index,y_index,z_index)
z0 = Kz(x_index,y_index,z_index)

% Filter the data around the center frequency
tau = 0.2;
filter = exp(-tau*((Kx-x0).^2+(Ky-y0).^2+(Kz-z0).^2)); % the Gaussian function
path = zeros(3,49);
for i = 1:49
    rawData = reshape(subdata(:,i),n,n,n);
    utn = fftshift(fftn(rawData));
    unft = filter.*utn; % filter the data with the defined Gaussian function
    unf = ifftn(fftshift(unft)); % transform back to the spatial domain
    maxf = max(abs(unf),[],'all'); % Determine the locations of maximum frequency (the submarine)
    [xPath,yPath,zPath] = ind2sub(size(unf), find(abs(unf) == maxf));
    path(:,i) = [X(xPath,yPath,zPath), Y(xPath,yPath,zPath), Z(xPath,yPath,zPath)];
end

% Plot the trajectory of the submarine
plot3(path(1,:),path(2,:),path(3,:),'-o','Linewidth',1); grid on
set(gca, 'FontSize', 12)
title('Path of the submarine over time')
xlabel('x');
ylabel('y');
zlabel('depth');
axis([-10 10 -10 10 -10 10])
path(:,end) % the final location of the submarine
```

Listing 1: MATLAB code.