COMPUTATIONAL METHODS FOR DATA ANALYSIS FINDING SUBMARINES

AVERY LEE

Applied Mathematics Department, University of Washington, Seattle, WA
leeave22@uw.edu
01/28/2022

Abstract. Finding Submarines produces the 3D path of the submarine given its noisy acoustic data over 24 hours in 30-minute increments. Through Fourier Transform, averaging, and Gaussian Filtering calculations, it denoises the data to accurately find the path. This paper will go through the mathematical theories behind the calculations, the algorithm used to find the 2D (bird's eye view) and 3D path over 24 hours, the final computational results, and conclusions.

1. Introduction and Overview

Signal processing can be used in various applications, including telecommunications, digital transmission, image processing, voice data processing, or sonar [1]. This paper will cover an example of 3D acoustic wave processing, where noisy data is denoised by 3D Fourier Transformations and Gaussian Filtering.

In this context, there is a submarine in the Puget Sound area emitting an unknown noisy acoustic frequency created by new technology. The submarine is constantly moving and the frequency emitted is noisy, so it can't be located easily. Our job is to locate its path given the broad spectrum recording of acoustics data obtained over 24 hours in 30-minute increments with the techniques mentioned above.

Section 2 of this paper will go over the theoretical mathematical background necessary to understand the computations and code. Section 3 will cover the algorithms and Python software packages. Then Section 4 will summarize the results from the computations, and Section 5 will summarize the learnings.

2. Theoretical Background

2.1. **Fourier Transform.**

A Fourier Transform (FT) can be used to break a signal down into frequencies. A Fourier Transform of f is often written as $\hat{f}$ and is a complex-valued function. The equation is written below, where e^{-ikx} represents the Fourier kernel [2].

$$F(f) = \hat{f}(\xi) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-i\xi x} dx$$

The Fourier kernel follows the Euler's equation, so

$$e^{-i\xi x} = cos(\xi x) + isin(\xi x)$$

An inverse to this transformation is also possible; in other words, turn $\hat{f}$ into f, with this equation below.

$$F^{-1}(\hat{f}) = f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\xi) e^{i\xi x} d\xi$$

One can see that the Inverse Fourier Transform of the Fourier Transform is f, or $F^{-1}(F(f)) = f$. This is called the Fourier Integral Theorem.

A Discrete Fourier Transform (DFT) basically conducts Fourier Transform over a finite interval given equally spaced data points, with the given equation below.

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

A DFT extracts the frequency components present in a signal. A drawback is that it studies the entire frequency at once, so it does not characterize changes in the frequency over time. Also, the algorithm has time complexity of $O(N^2)$, so it can take a long time to process when there is an extremely large dataset.

The Fast Fourier Transform is one of the most important algorithms in modern computing. It has $O(NlogN)$ time complexity, which is way more efficient than a DFT's $O(N^2)$ when it comes to large datasets. Furthermore, it produces very accurate results.

It is similar to a DFT, except it splits the DFT, each of length $\frac{N}{2}$. When this is repeated over and over again, it will finalize the transformation.

2.2. **Averaging.**

The given datapoints of frequency are separated into 49 sections, starting from the initial time to 48 more increments which each represent 30 minutes, for a total of 24 hours. At each time, different amount of frequencies exist in 3D space (represented as x,y,z). Not all the frequencies seen in this space and time is significant, in fact, there is a lot of white noise in the data, which is a normally distributed random variable with mean of 0. This means that the average of white noise is 0.

Averaging the frequencies at each space for all times will ultimately cancel out the noise, and the highest average gives the central frequency (or frequency signature). It is important to note that the average is taken over the frequency domain and not the time domain.

2.3. **Gaussian Filtering.**

There are various filtering methods but the most common is the Gaussian Filter, which was suggested by Gabor. Filtering is used in FFT graphing to select frequency components from a signal and filter out noise, especially when a central frequency is given. This filter would be multiplied to the FFT'ed signal in the frequency domain [3].

In this example, this 3D Gaussian filter is used. The a, b, and c are the central frequencies found. As discussed later in Section 3 and 4, there are 2 central frequencies found, which are $\pm a, \pm b, \pm c$, so both are taken into consideration.

$$g(x, y, z, a, b, c, \sigma) = e^{-\frac{(x-a)^2+(y-b)^2+(z-c)^2}{2\sigma^2}} + e^{-\frac{(x+a)^2+(y+b)^2+(z+c)^2}{2\sigma^2}}$$

3. Algorithm Implementation and Development

3.1. **Python Packages and Notable Functions.**

Python was used to compute the algorithms and produce plots. Below is a list of packages used.

`google.colab, drive`: Gives access to the user's Google Drive account where they can access the data.

`numpy`: Allows usage of multidimensional arrays and functions to manipulate those arrays. [4]

`pandas`: Data analysis library that is great for data analysis and manipulation of data. [5]

`plotly`: Plots high quality graphs, including 3D graphics where the user can control size, direction, and angle. [6]

`IPython.display`: Helps with figure and plot managing. [7]

`matplotlib.pyplot`: Useful for graphics manipulation. [8]

`numpy.meshgrid()`: Generates a 3D grid with given x,y,z points.

`numpy.fft.fftn()`: Computes an N-dimensional Fourier Transform.

`numpy.fft.ifftn()`: Computes an inverse N-dimensional FFT.

`numpy.fft.fftshift()`: Shifts the array so that the zero-frequency component is shifted to the center of the array.

`plotly.Isosurface()`: Produces a 3D plot given x, y, z ranges and data values.

3.2. **Setup.**

Now, this section will go over the detailed steps of the algorithm to output the submarine path.

The given data is a 64^3 x 49 matrix, where the rows represent a 3D area with $2^n = 64$ units on each side where n represents Fourier modes, and columns represent a 30-minute timestamp in the 24-hour period starting with initial time. In a way this is a 4D table with the 4th dimension being time. The length of the spatial domain is set to $L = 10$ for each side, but the frequency domain will need to be rescaled by $\frac{2\pi}{L}$ because there is an assumption of 2π periodic signals from $(-\pi, \pi)$. Now we have the spatial domain and frequency domain that can be plotted.

3.3. **Averaging.**

Now that the fundamentals are set up, the averaging technique mentioned above is used to find the central frequency. By iterating over all of the columns (time) of the original data, each 3D signal given from the original data is Fast Fourier Transformed, then FFT shifted so that the zero-frequency component is shifted to the center of the array. Then the sum of each frequency in the frequency domain in all 49 timestamps is found.

The average of the absolute value of each sum is found, where the maximum value is the central frequency. After finding the index of this maximum frequency, the x, y, z coordinates of the frequency domain can be found where the central frequency exists. These coordinates can be plotted to find 2 main points. More details about these 2 points are in Section 4.

3.4. **Filtering.**

Next, the central frequency is used to center the Gaussian filter, of which the equation is stated in Section 2. The x, y, z parameters are the frequency domain coordinates (often noted as kx, ky, kz), and a, b, c parameters are the central frequency coordinates found from averaging. A sigma of 2.5 was used, although it can be another value. After trial and error, 2.5 seemed to be an appropriate value. If the sigma is too large, it will not filter out the noise to find the submarine path. This filter is then shifted since it was shifted to find the central frequency earlier.

3.5. **Plotting Submarine Path.**

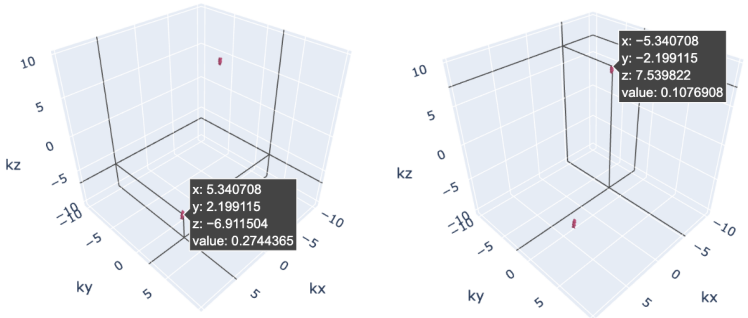
To plot both the 2D and 3D paths, the x, y, z coordinates need to be found over the 24-hour period at each 30-minute timestamp. Iterating over the timestamps (columns in original data), the FFT of each signal is multiplied by the filter, then inverse FFT'ed. After going through the filter, the data should have been denoised. As mentioned above, the FFTs are complex valued functions, so only the real values are pulled out. This is the denoised values to work with.

The indices of the maximum at each timestamp is found to find the coordinates of the submarine in the spatial domain. These coordinates of the submarine in the spatial domain are compiled at each iteration to find the final path of the submarine. To plot the 2D graph from bird's eye view, just plot the x and y coordinates. To get the 3D graph, plot the x, y, and z coordinates.

4. Computational Results

4.1. **Central Frequency.**

The central frequency calculated from the averaging method was (5.34, 2.2, -6.91). But by plotting, these values can be visualized.



(A) (B)

Figure 1. Central Frequencies in the Frequency Domain

From the coordinates displayed in Figure 1, we can see that both coordinates are approximately $(\pm 5.34, \pm 2.2, \pm 6.91)$. So we can use $(5.34, 2.2, -6.91)$ and $(-5.34, -2.2, 6.91)$ to use in the Gaussian Filter. To clarify, this will be the filter before the shift, where kx,ky,kz are in the frequency domain, and cfx, cfy, cfz are the central frequency coordinates. Again, $\sigma = 2.5$.

$$g(kx, ky, kz, cfx, cfy, cfz, \sigma) = e^{-\frac{(kx-5.34)^2+(ky-2.2)^2+(kz+6.91)^2}{2(2.5)^2}} + e^{-\frac{(kx+5.34)^2+(ky+2.2)^2+(kz-6.91)^2}{2(2.5)^2}}$$

4.2. **Submarine Path.**

From using the algorithms and calculations described in Section 3, these 3D and 2D (bird's eye view) plots can be created.

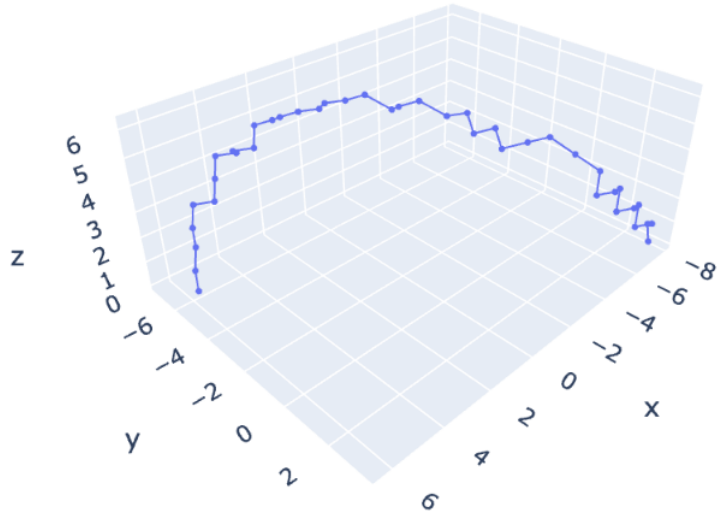


FIGURE 2. Submarine's 3D Path (24 hours, 30-minute increments)

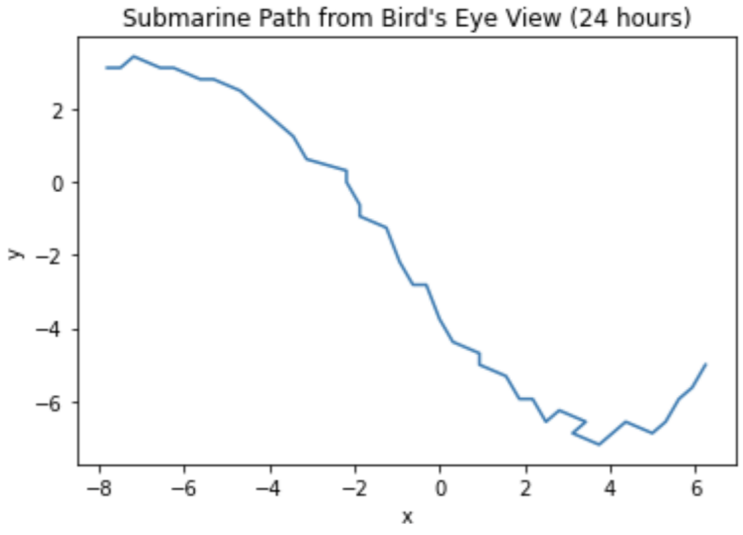


FIGURE 3. Submarine's Path from Bird's Eye View (24 hours, 30-minute increments)

For both Figure 2 and Figure 3, note that this is in the spatial domain, and the x axes do not represent time. When Figure 2 is seen from above, the result is Figure 3

Since it's not necessarily clear where the submarine moves from and to, the exact coordinates are shown in the table below. For clarity, the columns represent the x, y, and z coordinates, and the rows represent time.

	x	y	z
0	-7.813	3.125	0.000
2	-6.563	3.125	1.250
4	-5.625	2.813	3.125
6	-4.688	2.500	4.063
8	-3.438	1.250	5.313
10	-1.875	-0.625	5.313
12	-0.938	-2.188	6.250
14	0.313	-4.375	5.938
16	1.563	-5.313	5.625
18	2.813	-6.250	4.375
20	4.063	-6.875	3.438
22	5.313	-6.563	2.188
24	6.250	-5.000	0.938

TABLE 1. Submarine Location Every 2 Hours

Comparing the datapoints in Table 1 to Figure 3, it can be seen that the submarine is going from "$left$" to "$right$".

5. Summary and Conclusions

In this paper, Fourier Transform techniques (DFT, FFT, IFFT, FFTSHIFT), averaging to find central frequency, filtering (Gaussian filtering for this report), and 2D and 3D plotting were used to find the path of a submarine in a 24-hour period. Signal processing methods such as this can be used in a variety of situations. For this example specifically, this algorithm can be used to study the submarine's movements for longer amounts of time or shorter increments, and predict what the submarine is up to or its future movements. If another unknown signal appears in the Puget Sound one day, this algorithm, or a slight variation of it, can study and predict its movement.

Acknowledgements

The author is thankful to TA Katherine Owens for the general rundown and clarifications on calculations and code, as well as classmates such as Pranav Natarajan and Brian Powers that reconfirmed my data results and Python syntax. Last but not least, the author is thankful to Professor Bamdad Hosseini for teaching most of the mathematical concepts mentioned in the paper.

References

[1] A. V. Oppenheim. *Applications of digital signal processing*. 1978.
[2] Bamdad Hosseini. Signal processing with dft. University of Washington (LOW 216), Jan 2022.
[3] Bamdad Hosseini. Signal processing with dft. University of Washington (LOW 216), Jan 2022.
[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
[5] The pandas development team. pandas-dev/pandas: Pandas, February 2020.
[6] Plotly Technologies Inc. Collaborative data science, 2015.
[7] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.
[8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.