AMATH 582: ASSIGNMENT 1

JON STAGGS

Applied Mathematics M.S. Student, University of Washington, Seattle, WA
jrstaggs@uw.edu

Abstract. In this paper we are given noisy acoustic pressure data of a given region in the Puget Sound where a submarine is meandering and emitting an unknown frequency. We leverage properties of Fourier Series and construct a Gaussian filter to reduce the white noise in the data allowing us to identify the submarine's frequency and reconstruct an approximate path over a 24 hour time interval.

1. Introduction and Overview

We are provided noisy acoustic pressure measurements in a cube with time measurements every half hour for 24 hours. In this cube there is submarine emitting an unknown frequency. We will use a time average filter to determine the unknown frequency and construct a Gaussian filter to clean the noisy signal and recover the trajectory of the submarine. The techniques we use are based on the Fourier Transform and Fourier Series. The Fourier Transform allows us to analyze our signal in the frequency domain rather than in the time domain since the Fourier Transform gives you the amplitudes of the frequencies that occurred in the time domain [7]. We also have the Fourier Inversion Theorem which allows us to recover our original function. These tools will allow us to take our noisy signal and construct a path that approximates the trajectory of the submarine through time. In Sec. 2 we will provide further detail on Fourier Series and its discrete version. We use Python with the NumPy library to computationally approach this problem and Sec. 3 will outline our algorithm. In Sec. 4 we present location and amplitude of the dominant frequency as well as present the submarine's trajectory from multiple perspectives.

2. Theoretical Background

The main tool that we use is the Fast Fourier Transform so will give a brief overview of the connections between the Fourier Series, Discrete Fourier Transform, and the Fast Fourier Transform then move to discussing some theory of filtering. The bulk of the discussion regarding all things Fourier come from [4].

The Fourier Series is a series representation of a function $f : [0, 2L] \to \mathbb{R}$ that works by breaking $f(x)$ into simpler components, sine and cosine in this case, and the weights of these components are found by projecting the function f onto a basis made of sine and cosine. Since we use the 3D version we will present the 3D forms. Take $\vec{x} \in \mathbb{R}^3$ and $\vec{r} = (j, k, l)$ (i.e. the vector of indices) with the typical vector inner product $\langle \vec{x}, \vec{y} \rangle = \vec{y}^T \vec{x}$. For clarification we write $i = \sqrt{-1}$ and do not take it to be an index. We then have,

$$f(\vec{x}) = \sum_{l=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} c_{j,k,l} e^{\frac{i\pi}{L}[\langle \vec{r}, \vec{x} \rangle]}, \ \ c_{j,k,l} \in \mathbb{C}^3. \tag{1}$$

Date: January 27, 2022.

We can explicitly compute $c_{j,k,l}$ by the projecting f onto sine and cosine as mentioned above with the L_2 inner product. Here we will tighten up the notation by denoting a cube of side length $2L$ as $[0,2L]^3$ and using Euler's formula, $e^{i\theta} = \cos(\theta) + i\sin(\theta)$. The Fourier coefficients are,

$$c_{j,k,l} = \frac{1}{(2L)^3}\int_{[0,2L]^3} f(\vec{x})e^{-\frac{i\pi}{L}(\langle\vec{r},\vec{x}\rangle)}\mathrm{d}\vec{x} \tag{2}$$

To obtain the Discrete Fourier Series of $f(\vec{x})$ we truncate (1) with N points which yields,

$$f(\vec{x}) \approx \sum_{j,k,l=N/2}^{N/2-1} c_{j,k,l}e^{\frac{i\pi}{L}[\langle\vec{r},\vec{x}\rangle]} \tag{3}$$

with $c_{j,k,l}$ defined in (2). We are taking the convention from class that the Discrete Fourier Transform is $\hat{f} = (c_{(N/2,N/2,N/2)}, ..., c_{(N/2-1,N/2-1,N/2-1)}) \in \mathbb{C}^{3N}$, where $c_{i,j,k}$ are Fourier coefficients. The Fast Fourier Transform computes these coefficients in an efficient way. Two important notes about the Fast Fourier transform are 1) it assumes your function is periodic (because of e^{inx}), and 2) to have $\mathcal{O}(n\log(n))$ operation count your discretization has to have 2^n points [6]. If we have $\hat{f}$ and would like to return to our original domain then we take the inverse transform. The inverse is constructed by taking the Fourier Coefficients (2) and that we have calculated and rebuilding the sum (3) by multiplying by the corresponding phases, i.e. the exponential [10].

Our first filter leverages that adding mean zero white noise to a signal is equivalent to adding mean zero white noise to its Fourier Coefficient [3]. This allows us to create a filter by simply averaging the frequencies of our signal across time. Let $\hat{f}_n$ denote the 3 dimensional Discrete Fourier Transform at time $t = n$ then our time average Discrete Fourier Transform can be readily found by,

$$\hat{f}\text{avg} = \frac{1}{T}\sum_{n=0}^{T}\hat{f}_n. \tag{4}$$

Since we are assuming the noise is mean zero then by taking the time average of our frequencies we reduce the white noise. We now make a critical assumption, the only two frequency sources are the mean zero noise and the submarine. This allows us to now take the largest amplitude of (4) and identify it as the frequency emitted by the submarine. If we do not have this assumption nothing is stopping us from identifying a whale song (frequency range of 30Hz to 8kHz) or a dolphin whistle (frequency range of 20Hz to 30kHz) as the submarine's frequency if we simply take the largest frequency [8]. If the author was more versed on the native marine animals of the Puget Sound then we may be able to eliminate this assumption. Also observe that since f is a real valued frequency, we will have a symmetric signal in our Fourier domain.

As pointed out in [3] we only have limited time data so we use a filter to further eliminate noise. Filtering the signal, f, works by convolving the signal with another function, g, which we call a filter. Convolving two functions can be thought of as smoothing one function by another and can be helpful to think of it as a type of averaging. In that way, we are averaging two functions at each point across our domain and this is why filters tend to be compactly supported functions or functions of rapid decay. We can center our filter at some frequency so that this frequency and those near it are preserved while the compactly supported/rapid decay part of the filter function dampens frequencies further from the center.

It is often convenient to stay in one domain as long as possible so we use the fact that Discrete Fourier Transform of the convolution of functions is equivalent to multiplication of the Discrete Fourier Transforms of functions [4],

$$(\widehat{f * g})_k = 2\pi \hat{f}_k \hat{g}_k. \tag{5}$$

In this application, we use a sum of Gaussians centered at the 2 symmetric locations of the dominant frequency as our filter function. We use this filter because it is easy to construct and by finding our dominant frequency with the averaging process we know exactly where to center our filter. We now must choose an appropriate standard deviation. Consider convolving our signal with a dirac delta distribution,

$$(f * \delta)(x) = \int_{\mathbb{R}} f(x)\delta(x - \bar{x})\mathrm{d}\bar{x} = f(\bar{x}) \tag{6}$$

We see that if we choose a very narrow Gaussian we would only pick up a small band of frequencies and dampen all others out which may amplify noise near our center frequency. On the other hand, if we choose a larger standard deviation then we may not be able to dampen out enough of the erroneous frequencies and still retain noise.

3. Algorithm Implementation and Development

We utilized Python 3.8 with the NumPy and matplotlib packages and Spyder as our IDE [11],[1],[5],[9]. We began by following the Helper code [2] and set up our computational domain with *meshgrid* which yields the spatial domain $[-10, 10] \times [-10, 10] \times [-10, 10]$ and $N = 64$ grid points and our corresponding frequency grid is $[-10, 10] \times [-10, 10] \times [-10, 10]$. Before proceeding we make a remark on the *fft, fftshift, ifft,* and *ifftshift.* The convention in NumPy is to order the frequency output of the *fft* like,

$$k \in (0, 1, ...N/2 - 1, -N/2, -N/2 + 1, ..., -1). \tag{7}$$

The purpose of *fftshift* is to order the Fourier Coefficients in the usual way we'd expect and *ifftshift* puts the Fourier Coefficients back into the form (7) so NumPy can perform *ifft*, respectively. The procedure is *fftshfit*(*fft*(f)). If we have $\hat{f}$ then to shift back, *ifft*(*ifftshift*($\hat{f}$))[4]. In this section if we say "take the (Inverse) Fast Fourier Transform" we mean to apply the appropriate shifting procedure above and perform the transform so we will not mention shifting each time we say a transform is taken. We also often take the real part after inverse transforming due to the fact that round-off errors may add small imaginary components to our real valued signal.

To compute the time average Discrete Fourier Transform we build a *for* loop that is indexed by the discrete time steps. At each step in the loop we 1) reshape our data into an array of shape $(64, 64, 64)$ at the corresponding time step t, 2) take the 3D Fast Fourier Transform of our array, and 3) add this frequency to the previously computed frequency. Outside of the *for* loop we take the average and take the absolute value of this average. This allows us to extract the indices of the dominant (i.e. largest) frequency utilizing NumPy's *unravel_index* and the *argmax* method. Since we have a real signal we will have another frequency of approximately the same value and we can leverage symmetry about the index $(32, 32, 32)$ to easily find the second set of indices.

We can now construct a sum of Gaussian functions in the frequency domain that is centered on the frequency grid points corresponding to the 2 sets of indices previously found (see (8) for sneak peak).

To extract the coordinates of the trajectory we build a *for* loop that is indexed by the discrete time steps. At each step in the loop we 1) reshape our data into an array of shape $(64, 64, 64)$ at the corresponding time step t, 2) take the 3D Fast Fourier Transform of our array, 3) multiply the transformed array by our Gaussian filter, 4) take the Inverse Fast Fourier Transform of the product and take only the real part, and 5) extract the indices corresponding to the largest value of our acoustic pressure data using NumPy's *unravel_index* and the *argmax* method. Outside of the *for* loop we take these these indices and obtain the spatial coordinates corresponding to these indices which yields the trajectory of the submarine across the entire time interval.

4. Computational Results

We denote the frequency domain in wavenumber/Fourier modes (k_x, k_y, k_z) and amplitude is in units of Hurtz, Hz. In Fig. 1 we plot two (k_x, k_y) slices of the time averaged frequency (4) which show the locations and the amplitude of the dominant frequency. As noted in Sec. (2) we expected to see the same dominant frequency at two symmetric locations. The dominant frequency is located at approximately $(-2.1991, -5.3407, 6.9115)$ and $(2.1991, 5.3407, -6.9115)$ with an amplitude of approximately 89.8965 Hz.

This information allows us to construct our Gaussian filter centered at these locations in the Fourier domain with standard deviation σ. We have the filter,

(8)
$$G(k_x, k_y, k_z) = \frac{1}{\sigma\sqrt{2\pi}}\left(e^{-\frac{(k_x+2.1991)^2+(k_y+5.3407)^2+(k_z-6.9115)^2}{2\sigma^2}} + e^{-\frac{(k_x-2.1991)^2+(k_y-5.3407)^2+(k_z+6.9115)^2}{2\sigma^2}}\right).$$

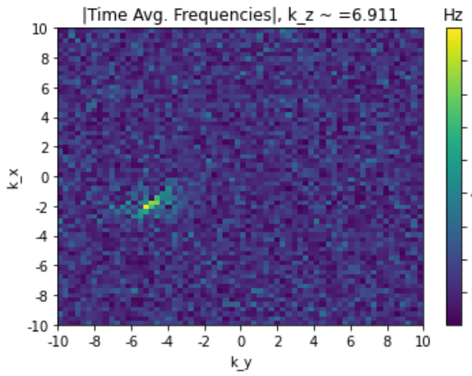
With our filter we are able to reconstruct the submarine's trajectory in spatial coordinates through time, however, as discussed at the end of Sec. 2 we will need to determine an appropriate σ for our filter to obtain a smoother trajectory. We visually inspect our trajectory with multiple values of σ which can be seen in Fig. 2. We can see that there is a Goldilocks situation that occurs when choosing our standard deviation which was expected as we saw in Sec. 2. For $\sigma = 1$ we see that in Fig.2a there is no coherent trajectory. We are simply not picking up broad enough frequency information so we isolating and amplifying high frequency noise along with our dominant frequency. For $\sigma = 27$ we see in Fig. 2c that we still have a fairly smooth trajectory except at a handful of points. While we are centered at the dominant frequency, we have a large standard deviation so our filter will only be able to dampen out frequencies that are massive in magnitude (a low probability event) so we end up carrying around frequencies that come from the white noise which are probabilistically more likely to occur in our (frequency) region of interest. Hence we settle on $\sigma = 7$ for our filter - see 2b. When testing out various values of σ we obtain approximately the same smooth trajectory for $3 \leq \sigma \leq 15$.

In Fig. 3 we present the submarine trajectory from 2 different azimuth angles. We can see roughly a parabolic shape in the z-coordinate. While Fig. 3a and Fig. 3b give us a better perspective of the y-coordinate and x-coordinate, respectively, we can do more. To better see the individual coordinates we plot each coordinate parameterized by time, Fig. 4. This also gives use information about the starting and ending location of the submarine in Fig. 3. First we note that in Fig. 4a we have that $x(t = 0) \approx 2$ and from Fig. 4b we have $y(t = 0) \approx -8$. So in Fig. 3a our starting point corresponds to the left-most y value and in Fig. 3b we start at the left most x value. In order to provide our special submarine tracking aircraft with a trajectory and starting location we can provide it with the y vs. x coordinates seen in Fig. 5 and using the information we found we determine the starting point of the $(x(t), y(t))$ trajectory to be coordinates in the bottom right of the figure. In a similar fashion we can determine the end point of the trajectory as $x \approx 4, y \approx 6$.

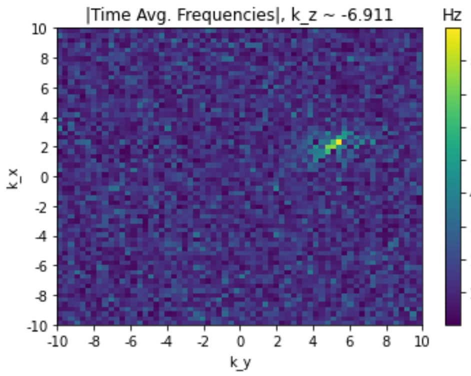
5. Summary and Conclusions

Using filtering techniques from signal processing we have taken noisy acoustic pressure data taken from a region in the Puget Sound where a wandering submarine is emitting an unknown frequency and identified the frequency as ≈ 89.8965 hz. With this we constructed a Gaussian filter with an appropriate standard deviation to trace the trajectory of the submarine over the past 24 hours.

Future directions that the author would like to explore are the robustness of this method by adding larger white noise- noise still with mean zero but much larger standard deviation, and looking at how one might find a center frequency and applying a filter function when the noise has non-zero mean or multiple noise sources that may or may not be mean zero.

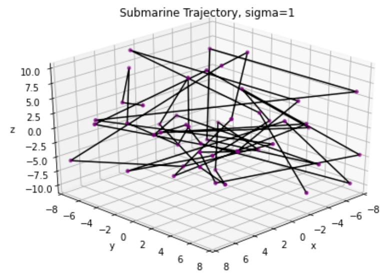


(A) $k_z \approx 6.911$

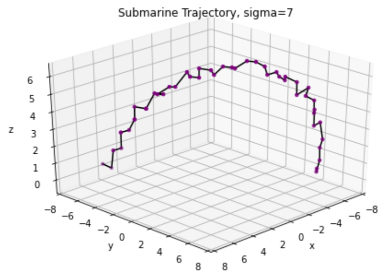


(B) $k_z \approx -6.911$

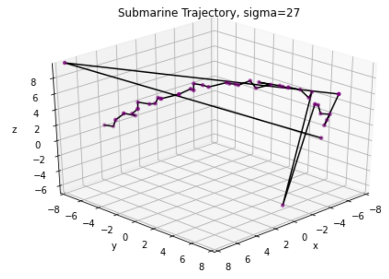
Figure 1. Symmetric dominant frequency locations in (k_x, k_y) plane at corresponding symmetric k_z of the time averaged frequency.



(A) $\sigma = 1$

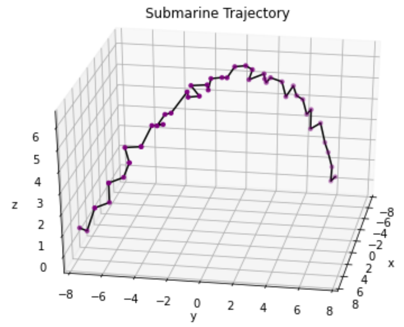


(B) $\sigma = 7$

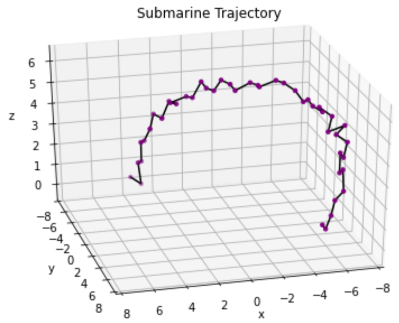


(C) $\sigma = 27$

Figure 2. Spatial trajectory of submarine after being filtered by Gaussian with 3 different standard deviations, σ.

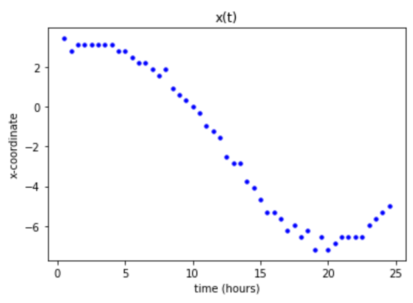


(A) $\theta = 10$

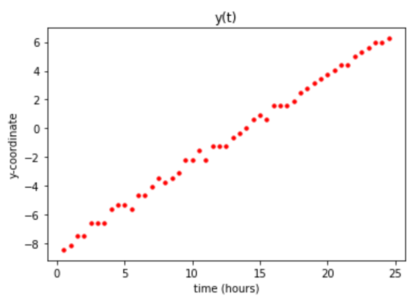


(B) $\theta = 75$

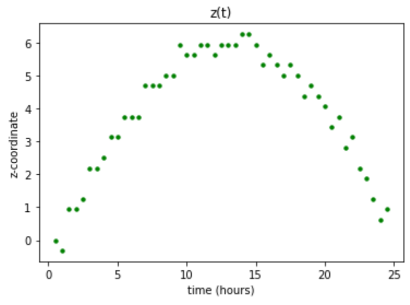
Figure 3. Spatial trajectory of Submarine at 2 different azimuth angles, θ, in degrees.



(A) $x(t)$



(B) $y(t)$



(C) $z(t)$

FIGURE 4. Trajectories of individual spatial coordinates parameterized by time.

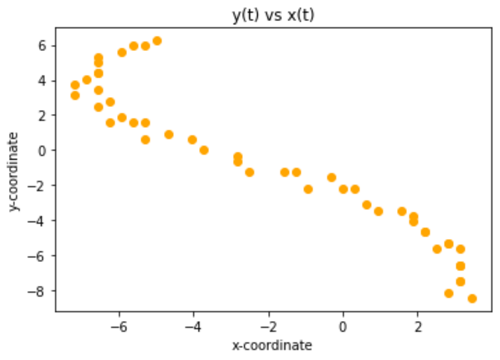


FIGURE 5. $(x(t), y(t))$ spatial coordinates of Submarine Trajectory.

ACKNOWLEDGEMENTS

The author is thankful to fellow classmate Daniel Leon for useful discussions about different technical assumptions that are implicitly being made as well as discussing the general flow of the algorithm. The author also thanks fellow classmate Ryan Mouw for pointing out NumPy's *unravel_index* function which made locating the dominant frequency and coordinates much easier. The author thanks Dr. Hosseini for the reminder that real signals give rise to symmetric frequencies giving rise to a sum of Gaussians for the appropriate filter function.

REFERENCES

[1] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.
[2] B. Hosseini. Amath 582 assignment 1 helper code. Code to start assignment 1 of Amath 582.
[3] B. Hosseini. Amath 582 assignment 1 sheet. Amath 582 Assignment 1 Sheet.
[4] B. Hosseini. Amath 582 course notes. Notes for Computational Methods for Data Analysis.
[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[6] J. N. Kutz. Amath 581: Practical scientific computing. Notes for Scientific Compution.
[7] J. N. Kutz. *Data-Driven Modeling and Scientific Computation*. Oxford University Press, 2013.
[8] K. Madin. Frequency ranges of marine animal sounds. *Oceanus*, 2012.
[9] P. Raybaut. Spyder-documentation. *Available online at: spyder-ide.org*, 2009.
[10] E. Stein and R. Shakarchi. *Fourier Analysis: An Introduction*. Princeton University Press, 2006.
[11] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.