AMATH 482: HOMEWORK 1

SOPHIA BARBER

Department of Applied Mathematics, University of Washington, Seattle, WA
sophiab3@uw.edu

Abstract. In this paper, we consider recorded acoustic pressure data obtained over a 24-hour period at 30-minute intervals. We aim to locate the path of a moving submarine, which emits an unknown acoustic frequency, over this time period. We detect the dominant frequency of the submarine by averaging the Fourier transform over all time steps. We use this to construct a filter to extract the center frequency in the Fourier domain, thereby cleaning the data. We then use the location of highest acoustic pressure in the clean data at each time step to determine the location of the submarine.

1. Introduction and Overview

In this problem, we have acoustic pressure data recorded over a 24-hour period at 30-minute intervals. Each measurement is 3-d and is taken on a uniform grid of size 64 x 64 x 64. The actual size of the spatial domain is 20 x 20 x 20. Our data has been put into a 262144 x 49 matrix. Each of the 49 columns corresponds to one time step, and is of length 262144, i.e. a flattened 64 x 64 x 64 3-d measurement.

Our goal is to determine the path (in 3-d space) of the submarine over this 24-hour interval. At any given time step, we expect the highest pressure reading to be at the location of the submarine at that time step. However, since our data is noisy, we cannot reliably say which pressure measurements are due to noise and which actually correspond to our submarine, so this will give an inaccurate measurement of the position of the submarine. So, we need to clean the data first in order to remove as much noise as we can before we can use it to accurately determine the position of the submarine. I will compare the path of the submarine extracted from the clean data vs. from the noisy data in Section 4.

In order to clean the data, we put the data into the frequency (Fourier) domain using the Fast Fourier Transform (Section 2). However, we don't know what the dominant frequency generated by the submarine is. So, we use techniques to extract the dominant frequency (Section 2) and then we use it to construct a 3-d Gaussian filter centered around the dominant frequency to clean the data. We can adjust the "width" of our filter to determine how much data we keep and how much we filter out and discard. A narrower filter filters out more data. Once we have applied the filter (Section 3), we can look at the clean data at each time step and reliably say that the location of highest pressure at that time step is also the location of the submarine, now that we have filtered out as much noise as possible. We use this to plot the 3-d trajectory of the submarine over time.

2. Theoretical Background

Discrete Fourier Transform:

The Discrete Fourier Transform (DFT) is a way of approximating the Fourier Transform(FT) when you only have measurements of your signal at a finite number of discrete points [1]. We will use the DFT to compute the Fourier Transform of our data since our measurements are over a 64 x 64 x 64 discrete grid.

Date: 01/26/2025.

Consider the 1-d case. Let our signal be represented by $f : [0, L] \to \mathbb{R}$, and assume that we only have measurements of our signal at N discrete points: $f(x_0), f(x_1), ..., f(x_{N-1})$. Then, from [1], the DFT (our approximation of the Fourier Transform) is given by

$$\hat{f}_k \approx \frac{1}{N} \sum_{n=0}^{N-1} f(x_n) e^{-2\pi i \frac{k_n n}{N}}. \tag{1}$$

Since we only have acoustic pressure measurements at a discrete number of points, we will use the DFT, generalized to 3-d, to obtain our Fourier Transform of the data. Computationally, we will use the Fast Fourier Transform (FFT) to compute the DFT. It is the same thing as the DDT, but while the DFT takes $\mathcal{O}(N^2)$ operations, the FFT uses symmetry to compute the DFT more efficiently and only takes $\mathcal{O}(N \log N)$ operations [1].

Extracting the Dominant Frequency by Averaging the Fourier Transform:

Before we can filter our data, we need to find the dominant frequency generated by the submarine. Looking at our data in the Fourier domain, the dominant frequency is the triple (k_{xc}, k_{yc}, k_{zc}) at which the Fourier coefficient of largest amplitude occurs. The presence of noise in our data may skew this calculation. However, if we average the Fourier coefficients over all of the time steps we have available to us, we find that since the noise has mean 0 in the Fourier domain, we can accurately identify the dominant frequency. I will explain:

Let X_t be a 64 x 64 x 64 tensor containing the clean data (in theory) at some time step t. Let $Y_t = X_t + \epsilon_t$ be the 64 x 64 x 64 tensor containing the noisy data, where ϵ_t is the noise component. We know that $\mathbb{E}[\epsilon_t] = 0$, as we are assuming that our noise is Gaussian noise with mean 0. In the Fourier domain, we have

$$\hat{Y}_t = \widehat{X_t + \epsilon_t} = \hat{X}_t + \hat{\epsilon}_t$$

due to the linearity of the Fourier transform. When we average over all time steps, we see that

$$\mathbb{E}[\hat{Y}_t] = \mathbb{E}[\hat{X}_t + \hat{\epsilon}_t] = \mathbb{E}[\hat{X}_t] + \mathbb{E}[\hat{\epsilon}_t] = \mathbb{E}[\hat{X}_t] + 0 = \mathbb{E}[\hat{X}_t],$$

where $\mathbb{E}[\hat{\epsilon}_t] = 0$ since adding mean 0 Gaussian noise to the signal is equivalent to adding mean 0 Gaussian noise to its Fourier coefficients [1]. This means that when we average over all time steps in the Fourier domain, the mean of the FT of the noisy data is the same as the mean of the FT of the clean data, and the noise disappears. Thus, if we find the dominant frequency of the Fourier coefficients averaged over all time steps, our calculation will be accurate and won't be obscured by noise.

Constructing a Gaussian Filter:

We will use a Gaussian filter to filter our data. A Gaussian filter is localized, so it keeps frequencies near its center and discards those far away. Since we will apply a Gaussian filter in the Fourier/frequency domain, we want to keep the frequencies closest to the dominant frequency, i.e. the ones that correspond to the submarine, and discard the others, i.e. the noise. That way when we put our data back into the spatial domain, we will have gotten rid of as much noise as possible.

We construct our Gaussian filter by creating a 3-d Gaussian function in our frequency domain that is discretized over a 64 x 64 x 64 grid. Our 3-d Gaussian function will have its center at (k_{xc}, k_{yc}, k_{zc}), i.e at our dominant frequency. The function is of the form

$$e^{-\tau((k_x - k_{xc})^2 + (k_y - k_{yc})^2 + (k_z - k_{zc})^2)} \tag{2}$$

where (k_{xc}, k_{yc}, k_{zc}) is our dominant frequency and (k_x, k_y, k_z) are the wave numbers k in the x, y, and z directions [1]. We discretize by creating a 3-d grid over which these wave numbers will vary.

The variable τ affects the "width" of the filter. In my filter, I use $\tau = 0.5$. If we make τ smaller (closer to 0), then our filter will be "wider", i.e. it will filter out and discard less data. If we make τ larger, then our filter will be "narrower", i.e. it will filter out and discard more data. We have

to use trial-and-error to see what τ is right for our particular data set so that our filter removes enough data that it eliminates the noise but doesn't get rid of too much of the critical data [1].

3. Algorithm Implementation and Development

Setup: Before we can begin implementing our algorithm, we must set up two grids, one for the spatial domain and one for the frequency domain. These are 3-d tensors since our problem is 3-dimensional. We do this using the `np.meshgrid` command [2]. As explained in Section , these grids are 64 x 64 x 64 with the spatial domain being of size 20 x 20 x 20, i.e. from -10 to 10 in each direction.

Task I: Identifying the Dominant Frequency

To filter our data, we must identify the dominant frequency generated by the submarine. This is explained in Section 2. To execute this computationally, at each time point we put our data into the Fourier domain using the `np.fft.fftn` command [2]. Then we must use the `np.fft.fftshift` command [2] in order to reorder the Fourier coefficients we get so that the indices of each Fourier coefficient match the indices of their corresponding frequency in the frequency grid of k-values that we set up using `np.meshgrid` [2]. We do this so that when we look at some Fourier coefficient and want to extract the frequency that corresponds to it, we can simply take the index of the Fourier coefficient and get the k-value that has that index, as our indices will line up. If we didn't shift, then the indices of the Fourier coefficients would not align with the indices of their corresponding frequencies/k-values in the grid of frequencies we set up.

As explained previously, at each of 49 time points we have a 64 x 64 x 64 tensor of data. So, we do the above process 49 times (once for each time step) and then add the Fourier coefficients up element-wise into a 64 x 64 x 64 tensor that contains the total. Then we divide the whole tensor by 49 to average over all time steps. As explained in Section 2, averaging the Fourier transform helps us deal with noise when extracting the dominant frequency.

We can use the code:

`np.unravel_index(np.argmax(avg_sig_hat), (64, 64, 64))` [2]

where `avg_sig_hat` is the 64 x 64 x 64 tensor of our Fourier coefficients averaged over all time steps, to extract the index of the Fourier coefficient of largest magnitude, i.e. the index of the Fourier coefficient that corresponds to the dominant frequency. Then we go to our grid of k-values and extract the frequency (k_{xc}, k_{yc}, k_{zc}) that has this index, which is our dominant frequency.

Task II: Constructing the Filter

Now that we have our dominant frequency, we can use it to construct a 3-d Gaussian function with its center at our dominant frequency. This is a function in the form of Equation (2), where (k_{xc}, k_{yc}, k_{zc}) is our dominant frequency and (k_x, k_y, k_z) are the x, y, and z components of the grid of k-values we created using the `np.meshgrid` [2] command. This means that our filter will also be a 64 x 64 x 64 tensor. The variable τ affects the "width" of the filter. In my code, I use $\tau = 0.5$. The larger we choose τ to be, the "narrower" our filter will be, i.e. we will filter out more data/noise. I arrived at this value of τ after trying several. In Section 4, I compare the differences in the submarine's trajectory with different values of τ.

Task II: Filtering the Data:

To actually filter the data, we transform it (at each time step) into the Fourier domain using `np.fft.fftn` and then `np.fft.fftshift` [2]. The Fourier-shifted data is still a 64 x 64 x 64 tensor. Then we multiply this element-wise by our Gaussian filter (also a 64 x 64 x 64 tensor), and the put it back into the signal (spatial) domain using the commands `np.fft.fftshift` and then `np.fft.ifftn` [2].

Then we can take the (x, y, z) location of the maximum acoustic pressure in our cleaned data at a time step to be the location of the submarine at that time step, and repeat for each time step.

Again, we extract the index of the (x, y, z) location of highest acoustic pressure in our 64 x 64 x 64 grid of cleaned data at a time step by using the `np.unravel_index` [2] command, and then get the actual (x, y, z) location by extracting the x, y, and z values from our spatial grid we created using the `np.meshgrid` [2] command that have this index.

Now that we have the (x, y, z) location of the submarine at each time step, we can plot it in 3-d space, or plot a 2-d view of its trajectory in the (x, y) plane (**Task III**).

4. Computational Results

Task I: After applying the algorithm outlined in Section 3 to find the dominant frequency, we find that it is $\mathbf{k}^* := (k_{xc}, k_{yc}, k_{zc}) = (5.340707511102648, 2.199114857512855, -6.911503837897545)$. We can see this visually in Figure 1 below. The purple dot is the location, in frequency space, of our dominant frequency $\mathbf{k}^*$. I have plotted this on an Isosurface plot [5, 4]. This means that the blue blob is the location in frequency space of the frequencies that correspond to the Fourier coefficients of largest magnitude (i.e. the dominant/central frequencies). We can see that this aligns with the location of the dominant frequency we found, as we would expect. (Note that the purple dot has been enlarged simply for ease of viewing. It is in reality just one point, not several).

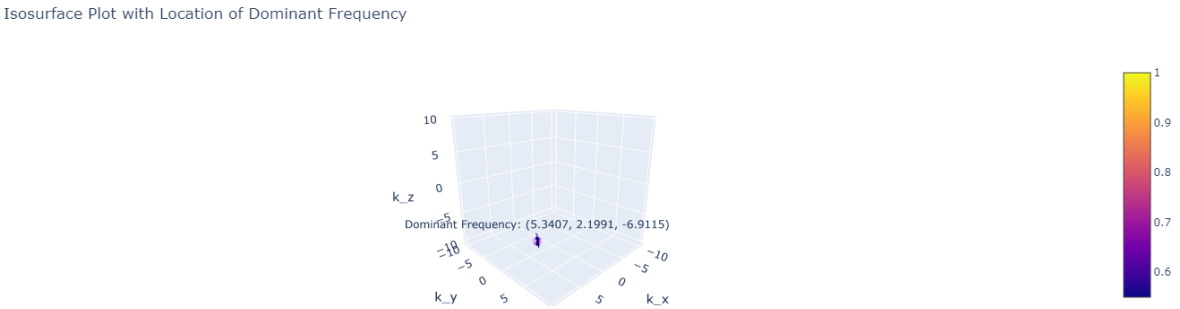
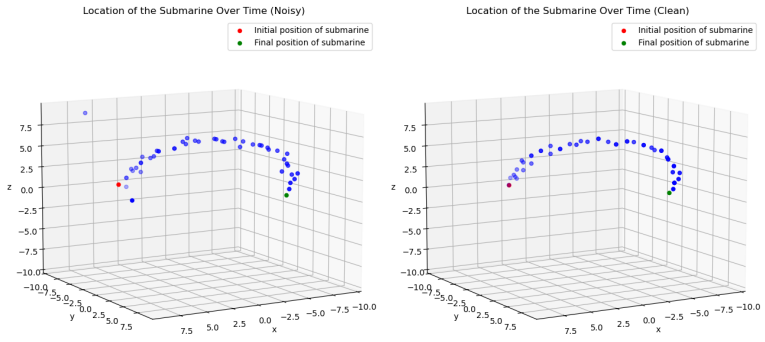


Figure 1. Isosurface plot showing the location of the central frequencies in Fourier space (dark blob). The calculated dominant frequency $\mathbf{k}^*$ is the light purple dot.

Task II: After filtering our data, we found the 3-d trajectory of the submarine using the algorithm in Section 3. In Figure 2 (next page), I show the trajectory extracted from the original, noisy data (2a) and the trajectory extracted from the filtered, clean data (2b) [3]. I used $\tau = 0.5$ in my filter for (2b) (Section 2). We see that the path extracted from the filtered data in Figure 2b is much smoother and less scattered than the path extracted from the noisy data in Figure 2a, and the outliers have been removed.

In Figure 3 (next page), we show what happens when we adjust τ, the "width" of our Gaussian filter [3]. We know that larger values of τ make the filter narrower, and filter out more data, whereas smaller values of τ make the filter wider, removing less data. After trying many values of τ, we found that the one that made the filtered submarine path the smoothest was $\tau = 0.5$ (see (3-C)). In (3-A), where $\tau = 3$, we see what happens when we choose a value of τ that is too large. We see that the path is actually more noisy than with $\tau = 0.5$, presumably because our filter is too narrow and some important data is being filtered out, leaving the path actually less smooth. In (3-B), where $\tau = 0.001$, we see what happens when we choose a value of τ that is too small. We see that the filter is so wide that it is practically filtering out no noise at all, as the path with $\tau = 0.001$ is barely better than the unfiltered, noisy path in (3-D). This shows how important it is to choose an appropriate value of τ, so that the filter is the right width for our data.



(A) (B)

FIGURE 2. 3-d trajectory of the submarine over time, extracted from the original, noisy data (A), vs from the filtered, clean data (B), where we used the value of $\tau = 0.5$ in our filter.

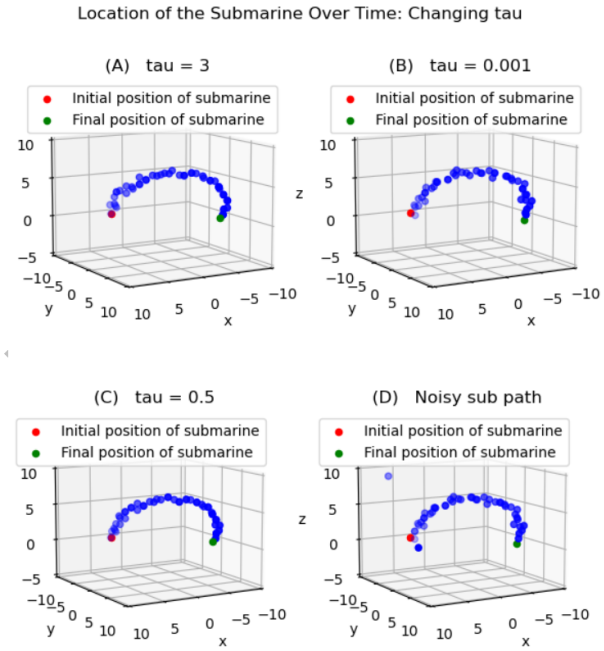


FIGURE 3. 3-d trajectory of the submarine, showing the effect of the choice of τ in the filter. (A) Uses $\tau = 3$ in the filter, (B) uses $\tau = 0.001$, (C) uses $\tau = 0.5$ (what we found to be optimal τ), and (D) is the original, noisy path.

Task III: In Figure 4 on the next page, we plot the (x, y) coordinates of the submarine's path over time [3]. Again, we can see that the filtered submarine path removes the outliers and is considerably smoother than the original path.

5. Summary and Conclusions

In this paper, we were able to identify the dominant frequency generated by the submarine through averaging of the Fourier transform (computed using FFT). We then constructed a Gaussian filter with its center at this dominant frequency. We filtered our data at each time step by applying

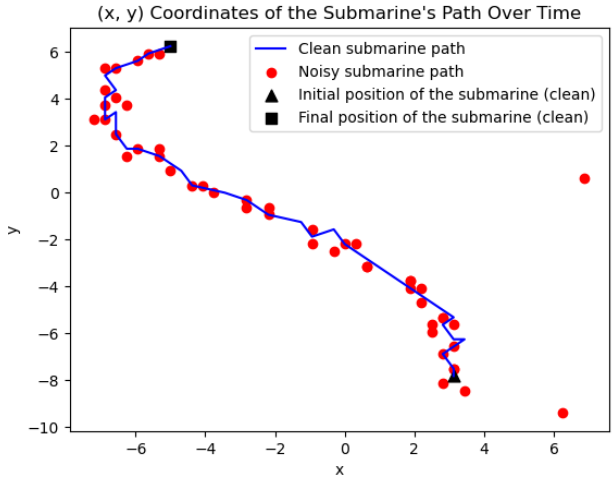


FIGURE 4. (x, y) coordinates of the submarine's path over time. The clean submarine path is marked by the blue line. The noisy, unfiltered submarine path is marked by the red dots. The black triangle denotes the submarine's initial position, and the black square denotes the final position (both from the clean path).

the Gaussian filter in the Fourier domain. We then identified the 3-d location of the submarine by looking at the location of maximum acoustic pressure in our cleaned data at each time step. This allowed us to visualize the 3-d path of the submarine over the 24-hour period. We found that the quality and smoothness of the path of the submarine extracted from the filtered data depended greatly on the width of our filter, which was dictated by the variable τ, and saw what happened when we chose different values of τ. By comparing our clean submarine path to the noisy submarine path, in both 3-d space and in the (x, y) plane, we were able to see that our filtering method removed much of the noise, making the path smoother and removing outliers. However, our path was still not super smooth so in the future we might look at other filtering methods and compare the results with those we obtained here with a Gaussian filter.

ACKNOWLEDGEMENTS

The author is thankful to Dr. Frank for her lectures on Fourier Transforms and Gaussian filtering methods, as well as for explaining the effects of changing the width of the filter, and in helping us to understand the concept of the dominant frequency. We are also thankful to Rohin Gilman for explaining why averaging the Fourier transform allows us to deal with the presence of noise in our data when identifying the dominant frequency, and for explaining the concepts behind this problem and helping with code implementation. Furthermore, our peer Eunice Han was helpful in implementing the algorithm for the extraction of the dominant frequency and in discussions of optimal filter width.

REFERENCES

[1] N. Frank. Amath 482 lecture notes. *Department of Applied Mathematics, University of Washington*, January 2025.
[2] C. R. Harris, K. J. Millman, S. J. van der Walt, et al. Array programming with NumPy. *Nature*, 585:357–362, 2020.
[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[4] F. Pérez and B. E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.
[5] Plotly Technologies Inc. Collaborative data science. 2015.