Fourier Analysis to Locate Moving Submarine from Noisy Data

Alex Mallen

Abstract

I will be demonstrating a technique to separate signal from noise by transforming data into the frequency domain. This process, the Fourier transform, allows one to remove noise from a signal even when the signal translates through the spatial domain. I apply this denoising technique to recover the 3D trajectory of a submarine from noisy data by finding the characteristic signal of the submarine and then filtering around that frequency.

1 Introduction

Noisy data frequently obscures underlying signals. Often, these patterns will be so obscured that they cannot be spotted by visual inspection of the data. Furthermore, high dimensionality can make such visual analyses challenging. One common technique to resolve this problem is to sample a large number of these events and compute the sample average. If the noise is additive, independent across realizations, and centered at 0, then averaging will eliminate this noise.

Unfortunately, many scenarios do not allow for multiple realizations of the same event. One such case is a constant signal that is moving through space. Our task is to locate the path of a submarine through 3D space based on acoustic data, but our data is noisy, so

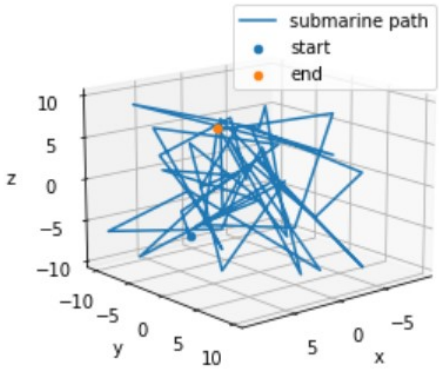

FIGURE 1: This is the path of the submarine constructed by naïvely selecting the location from each snapshot that produced the strongest acoustic signal. The noise is stronger than the signal, which prevents us from gaining insight into the location of the submarine directly.

it is not possible to pick out the submarine directly. A naïve inspection of the noisy data produces figure 1. We have one set of spatial measurements for every point in time and the submarine is moving along a yet-unknown path, which makes averaging the snapshots to cancel out the noise unhelpful.

To solve this, we compute the Fourier transform of the data before averaging, then transform the data back into the spatial domain. Because the signals will appear approximately the same in the frequency domain regardless of the location of the submarine, but the noise will still be random, averaging in the frequency domain can recover the signal.

Once the signal has been recovered, it is still necessary to recover the spatial origin of the signal. In our application to locating a submarine from noisy data, this step equates to finding the 3D coordinates of the submarine at each point in time. In order to recover the location of the submarine, we again look at the frequency domain, and we apply a Gaussian filter centered at the frequency.

2 Theory

2.1 The Fourier Transform

The Fourier transform assumes our data $\mathbf{x}$ is a vector of uniformly sampled points on a continuous function f that is periodic in 2π. The Fourier transform allows us to write our function f as a sum of complex exponentials as follows [1].

$$\mathbf{x}_t = \sum_{n=-\infty}^{\infty} \hat{\mathbf{x}}_n e^{nti} \tag{1}$$

where the Fourier coefficients $\hat{\mathbf{x}}_n$ are determined by

$$\hat{\mathbf{x}}_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{-nti} dt. \tag{2}$$

This integral computes the dot product between our function f and e^{-nti} for each n, which projects f onto a basis of complex exponentials.

In reality, however, the integral must be discretized so that it can be computed using only our sample of points $\mathbf{x}$. Likewise, we can only project onto finitely many basis functions. Therefore, the discrete Fourier transform (DFT) is used. The DFT computes the Fourier coefficients $\hat{\mathbf{x}}$ as

$$\hat{\mathbf{x}}_n = \sum_{k=0}^{N-1} \mathbf{x}_k e^{-k\frac{2n\pi}{N}i} \tag{3}$$

where N is the number of components in $\mathbf{x}$ [6]. The inverse DFT is similar:

$$\mathbf{x}_n = \frac{1}{N} \sum_{k=0}^{N-1} \hat{\mathbf{x}}_k e^{k\frac{2n\pi}{N}i} \tag{4}$$

Algorithm 1: Finding the wavenumbers k_x, k_y, and k_z with the maximum associated coefficient. The 3D FFT assumes the data lies in 2π cube, when our data actually came from a 2L cube, so we have to re-scale the wavenumbers by $2\pi/2L$ to get the true wavenumbers. Furthermore, the non-negative frequencies are shuffled to the beginning of each axis, which determines the arrangement of wavenumbers in k. For more details see `numpy.fft.fftn` of Appendix B.

import data from `subdata.csv` into `subdata`
initialize Fourier modes $k := \frac{2\pi}{2L} \cdot [0, 1, ...31, -31, ..., -1]$
initialize $X, \hat{X}$ to store spatial data, frequency data
for $t = 1 : t_{\max}$ **do**
  reshape `subdata[t]` into $64 \times 64 \times 64$ array and append it to X
  append `fft3`$(X[t])$ to $\hat{X}$
end for
$\hat{\mathbf{x}}_{\text{avg}} :=$ `sum`$(\hat{X})$ / $t_{\max}$
$i_{\max}, j_{\max}, l_{\max} :=$ `argmax`$(\hat{\mathbf{x}}_{\text{avg}})$
$k_x, k_y, k_z := k[i_{\max}], k[j_{\max}], k[l_{\max}]$

The Fourier coefficient $\hat{\mathbf{x}}_n$ represents how much of the frequency $\frac{2n\pi}{N}$ is present in the data $\mathbf{x}$, assuming our data $\mathbf{x}$ is sampled on an interval of length 2π. This also implies the assumption that f is periodic in 2π because the DFT assumes periodic boundary conditions. The DFT can also be computed for spatial domains of greater than one dimension and conveys the same information.

2.2 Denoising

We assume that our submarine data $\mathbf{x}$ consists of a localized signal $\mathbf{u}$ obscured by Gaussian noise $\mathbf{n} \sim \mathcal{N}_3(0, 1)$.

$$\mathbf{x} = \mathbf{u} + \mathbf{n} \tag{5}$$

The frequency domain therefore looks like $\hat{\mathbf{x}} = \hat{\mathbf{u}} + \hat{\mathbf{n}}$, where $\hat{\mathbf{u}}$ is localized and approximately constant for all snapshots, and $\hat{\mathbf{n}}$ is still independent random noise centered at 0. When $\hat{\mathbf{x}}$ is averaged over all snapshots, the noise will tend to 0, leaving the frequency signature $\hat{\mathbf{u}}$.

Once we know the frequency signature of the signal, we can construct a Gaussian kernel around that frequency. When we multiply a noisy signal by this filter in the frequency domain, it removes most of the frequencies that are present only because of noise. When transformed back to the spatial domain using the inverse FFT, the signal should be clearly localized and obscured by little noise.

3 Implementation

The first step to locate the submarine is to find the frequency signature of the submarine. The data is organized into $t_{\max} = 49$ snapshots from a 24 hour period, each of which has a

Algorithm 2: Finding the coordinates C of the submarine at each snapshot through filtering in the frequency domain. This is performed once k_x, k_y, and k_z are known through algorithm 1.

construct G, a Gaussian filter around (k_x, k_y, k_z)
$s := \frac{2L}{64} \cdot [-32, -31, ..., 31]$
initialize C, a list of submarine coordinates
for $t = 1 : t_{\max}$ **do**
  $\hat{\mathbf{u}} := G \odot \hat{X}[t]$ # apply filter element-wise to $\hat{X}[t]$
  $\mathbf{u} :=$ `ifft3`$(\hat{\mathbf{u}})$
  $\mathbf{c} := s[$`argmax`$(\mathbf{u})]$
  append $\mathbf{c}$ to C
end for

$64 \times 64 \times 64$ cube of noisy data representing how likely it is for the submarine to occupy that voxel. The cube has a sidelength of $2L = 20$.

The three wavenumbers k_x, k_y, and k_z are calculated by finding the argmax of the time-averaged DFT of the submarine data, as in algorithm 1. This is the frequency signature of the submarine. The specific implementation of the DFT I use is the fast Fourier transform (FFT) because it is accurate and reduces the time complexity of the DFT to $O(N \log(N))$. I used Python's numpy library implementations `fftn` and `ifftn`, which compute the n-dimensional FFT. Numpy recursively cuts the problem in half to exploit symmetries in the calculated terms of equation 3. Numpy implements `ifftn` in the same manner for equation 4. For more details see Appendix A and [3] and [4]. The argmax should be computed using the absolute value of the input. Numpy's `argmax` function otherwise discards the imaginary part.

To remove the noise within each snapshot a spherical Gaussian filter is multiplied by the FFT of that snapshot, which is then taken back into the spatial domain. This is outlined in algorithm 2. This process results in a list of coordinates C that describes the submarine's path through the water. The Python implementations of algorithms 1 and 2 can both be found in Appendix B, and on the GitHub repository linked in [5].

4 Results

The frequency signature of the submarine was found to be (2.20, 5.34, -6.91). These are the wavenumbers k_x, k_y, and k_z which appeared most dominant in the snapshot-averaged FFT.

The submarine's 3D path is shown in figure 2. It traveled in the positive-x negative-y direction for the majority of its trip. It starts off its trip moving upward until it reaches around 5.9 z then continues downwards. Figure 2 also shows the x and y coordinates of the path along which a tracking aircraft should be sent. A sampling of these coordinates can be found in table 1. The most up-to-date location of the submarine is (6.25, -5.0, 0.94).

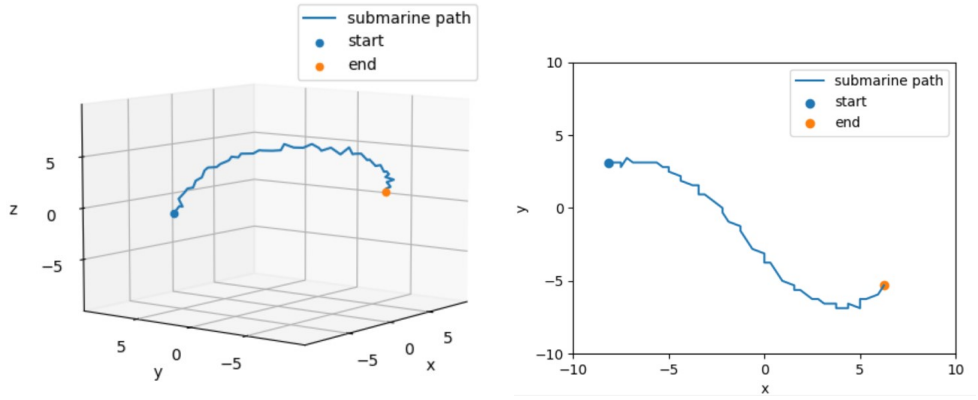

FIGURE 2: LEFT: This is the reconstructed path of the submarine from the acoustic data. This was obtained using a Gaussian filter with scale 3 (distance from center to inflection). Scales much less than 3 removed both the noise and the signal. Scales greater than 3 produced similar results until around 7, where little noise is filtered. RIGHT: This is the path of the submarine projected onto the xy-plane. This is the path along which a tracking aircraft should be sent.

time	**0:00**	**2:00**	**4:00**	**6:00**	**8:00**	**10:00**	**12:00**	**14:00**	**16:00**	**18:00**	**20:00**	**22:00**	**24:00**
x	-8.4	-6.9	-5.6	-4.7	-3.4	-1.6	-0.9	0.3	1.6	3.1	4.1	5.3	6.2
y	3.4	3.1	2.8	2.2	0.6	-0.6	-2.2	-4.1	-5.3	-7.2	-6.9	-6.2	-5.0
z	0.3	1.6	2.8	4.1	5.0	5.6	5.9	5.9	5.6	4.4	3.4	2.2	0.9

TABLE 1: A sampling of the recovered coordinates of the submarine over the 24 hour period.

5 Conclusion

This application demonstrates the utility of the Fourier transform in denoising moving signals to (a) find their frequency signature and (b) determine the spatial origin of the signal. Our original data was too noisy to determine the location of the submarine and too high-dimensional to make a visual guess. However, the frequency-domain denoising techniques were able to recover a trajectory of the submarine with relatively small error. The trajectory is smooth enough to fit a spline interpolation to further reduce the noise.

The capability to remove noise from moving signals comes from the fact that the Fourier transform of a signal is nearly invariant with respect to the location of the signal. However, this property also comes with significant drawbacks. Namely, the Fourier transform tells us very little about the location of a given frequency in the input space, which is important information for constructing audio spectrograms and compressing images, among many other applications [2].

Appendix A

Important functions and their implementations

`numpy.fft.fftn`

Takes an n-dimensional array of complex data as input and returns a complex array of the same shape representing how much of each frequency is present in the data. Assumes that the interval spanned by each axis is 2π and that the data is periodic in 2π along each axis.

The result has the 0 frequency of each axis at index 0, all the positive frequencies in the first half of each axis, and all the negative frequencies in the second half of each axis (for an axis of length m: $k = [0, 1, \ldots, m/2, -m/2, \ldots, -1]$). In order rearrange these Fourier coefficients such that the frequencies increase linearly from minimum to maximum, `fftshift` can be performed, which essentially shuffles the second half of each axis to the front of each axis. See [3].

`numpy.fft.ifftn`

Takes an n-dimensional array of complex data as input and returns the inverse of `fftn` such that `ifftn`(`fftn`(x)) = x.

`ifftn` expects that the input has the 0 frequency of each axis at index 0, all the positive frequencies in the first half of each axis, and all the negative frequencies in the second half of each axis. See [4].

Appendix B

Python code

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
from numpy.fft import fftn, ifftn, ifftshift
import matplotlib.pyplot as plt

subdata = np.load("subdata.npy", allow_pickle=True)

num_snapshots = subdata.shape[1] # 49
L = 10 # length of space interval [-L, L]
n = int(subdata.shape[0]**(1/3) + 0.5) # 64
s = np.linspace(-L, L, n, endpoint=False)
k_axes = ifftshift(2 * np.pi / (2 * L) * np.arange(-n // 2, n // 2)) #
    rescale wavenumbers bc fft assumes 2pi period

# ALGORITHM 1
# #--Averaging in frequency domain to reduce noise--#
X = [] # list of 64x64x64 arrays of spatial data in order from t=0:48
Xt = [] # frequency domain
for i in range(num_snapshots):
    x = np.reshape(subdata[:, i], (n, n, n))
    X.append(x)
    Xt.append(fftn(x))

xt_avg = sum(Xt) / num_snapshots


k_amax = np.unravel_index(np.argmax(abs(xt_avg)), xt_avg.shape) # indices
    of the maximum 3D frequencies
k_max = tuple(k_axes[i] for i in k_amax) # frequencies with max value
print("k_max:", k_max)

# ALGORITHM 2
# #--Gaussian Filter--#
scale = 3
A, B, C = np.meshgrid(k_axes, k_axes, k_axes) # 3D fourier space. cubes
    with x-freqs, y-freqs, and z-freqs respectively
kernel = np.exp(-1 / scale**2 * ((A - k_max[0])**2 + (B - k_max[1])**2 +
    (C - k_max[2])**2)) #

coords = np.zeros((num_snapshots, 3)) # array of 49 3D coordinates of the
    submarine
```

```
for i in range(num_snapshots):
    denoised_xt = Xt[i] * kernel
    denoised_x = ifftn(denoised_xt)
    idxs = np.unravel_index(np.argmax(abs(denoised_x)), denoised_x.shape)
    coords[i, :] = s[idxs[0]], s[idxs[1]], s[idxs[2]]

# 3D trajectory
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(coords[:, 0], coords[:, 1], coords[:, 2], label="submarine path")
ax.scatter(coords[0, 0], coords[0, 1], coords[0, 2], label="start")
ax.scatter(coords[-1, 0], coords[-1, 1], coords[-1, 2], label="end")
ax.view_init(elev=15, azim=50)
plt.xlabel("x")
plt.ylabel("y")
ax.set_zlabel("z")
ax.set_xlim(-10, 10)
ax.set_ylim(-10, 10)
ax.set_zlim(-10, 10)
ax.set_xticks(np.linspace(-10, 10, 5, endpoint=True))
ax.set_yticks(np.linspace(-10, 10, 5, endpoint=True))
ax.set_zticks(np.linspace(-10, 10, 5, endpoint=True))
plt.legend()
plt.show()

# 2D projection
fig = plt.figure()
ax = fig.gca()
plt.plot(coords[:, 0], coords[:, 1], label="submarine path")
plt.scatter(coords[0, 0], coords[0, 1], label="start")
plt.scatter(coords[-1, 0], coords[-1, 1], label="end")
plt.xlabel("x")
plt.ylabel("y")
ax.set_xlim(-10, 10)
ax.set_ylim(-10, 10)
ax.set_xticks(np.linspace(-10, 10, 5, endpoint=True))
ax.set_yticks(np.linspace(-10, 10, 5, endpoint=True))
plt.legend()
plt.show()

# for table. use https://www.tablesgenerator.com/#
for j in range(3):
    for i in range(0, 49, 4):
        print(round(coords[i, j], 1), end=" ")
    print()

for i in range(0, 25, 2):
```

```
    print(f"{i}:00", end="\t")

print("\nfinal coords:", coords[-1, 0], coords[-1, 1], coords[-1, 2])
```

References

[1] Kutz, Jose Nathan. time frequency 1.
https://www.youtube.com/watch?v=3kj8a-U5rZI

[2] Kutz, Jose Nathan. Time Frequency Analysis & Gabor Transforms.
https://www.youtube.com/watch?v=4WWvvMkFTw0

[3] Numpy fftn.
https://numpy.org/doc/stable/reference/generated/numpy.fft.fftn.html

[4] Numpy ifftn.
https://numpy.org/doc/stable/reference/generated/numpy.fft.ifftn.html

[5] Mallen, Alex Troy. GitHub source code.
https://github.com/AlexTMallen/AMATH-582/tree/master/HW1

[6] Wikipedia. Discrete Fourier Transform.
https://en.wikipedia.org/wiki/Discrete_Fourier_transform