AMATH 482 Homework 1
Hunting for a Submarine

Weien Guo

January 27, 2021

Abstract

A submarine with new technology emits noisy acoustic data. We are given a three-dimensional acoustic signals that is obtained over a 24-hour period in half-hour increments. We want to denoise the data and locate that submarine. By utilizing the Fast Fourier Transform (FFT) and other relating methods, we can reduce the noise in the data and retrieve clean, useful and accurate location coordinates. MATLAB has built-in functions that aid us in solving this problem. The functions along with their descriptions and the code can be found in the Appendix.

1 Introduction and Overview

The submarine with new technology is moving, so its location and path are unknown and need to be determined. A P-8 Poseidon subtracking aircraft is standing by, waiting for the trajectory and coordinates of the submarine. The given signal dataset is 262144*49, that is the data is collected at 49 time points, and each column can be transformed into 64*64*64 3-D data space. We assume all the noise in this problem is white noise, which has a normal distribution with zero mean and finite standard deviation.

In the process of removing noise, we would first need to average the data in the frequency domain. This reduces the effect of random noise in the data, and helps us to determine the signature frequency (center frequency) generated by the submarine. Then, we will apply a Gaussian filter, which is centered around this signature frequency, to each signal frequency to get the path of the submarine. At the end, the trajectory obtained will be visualized in a 3-D plot, and the coordinates will be sent to the subtracking aircraft.

2 Theoretical Background

2.1 Fourier Transform (FT)

Given a function, we can write it in terms of sines and cosines. *Fourier Transform* is a very useful tool to make noisy data interpretable. It takes in a function of space or time, and converts it into a function of frequency k. The Fourier Transform of $f(x)$, written $\hat{f}(x)$, is defined by the formula

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

The *Inverse Fourier Transform* recovers the original function, and is given by the formula

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

The above two formulas assume infinite domain. However, our problem has a finite domain and is discrete, so we consider the *Discrete Fourier Transform* (DFT) only. $\{x_0, x_1 ... x_{N-1}\}$ are equally spaced and is given by

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

When calculating DFT, the run time for its operation on N numbers is $\mathcal{O}(N^2)$. That is very slow when dealing with large datasets. The *Fast Fourier Transform* (FFT) can be referred as "divide and conquer algorithm" and do the calculation faster, with $\mathcal{O}(N \log N)$. The algorithm splits data into 2^n points and also assumes the user is on a 2π periodic domain, and shifts data, turning frequencies in the order

$$k = 0, 1, ..., \frac{N}{2} - 1, \frac{N}{2}, ..., -1$$

We can simply shift the frequency back by using the built-in MATLAB function `fftshift()`.

2.2 Averaging of the Spectrum

The real world data/signals are never clean and without any noise. We want to distinguish the true signal by reducing the noise. Since we have assumed that the noise has zero mean, we can take the average of realizations in the frequency domain and the noise will automatically reduced to close to 0. Then the frequency reveals itself and we can use it as the signature frequency in later parts. It is worth noting that although the signal comes at different time steps, the signal itself does not change in the frequency domain. So the changing location does not affect the summation of realizations in the frequency domain, and the noise can still be successfully reduced in this way. In other words, averaging would not work in the time domain, and averaging loses the time information in the data.

2.3 Spectral Filtering

After knowing the signature frequency, we can filter the data to further denoise. A one-dimensional *Gaussian* filter is given by

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

where τ is the width of the filter and k_0 indicates the center of the filter. A small τ would give a large window, and vice versa. Imagine a Gaussian bell curve, the filter would amplify frequencies close to the center and diminishes the rest that is farther away. Mathematically, we just multiply the frequency by the filter, and use the inverse Fast Fourier Transform to get the original signal and detect its location. If we center the filter around a wrong frequency, no signal will be detected. Since our problem is 3-D, the Gaussian filter we use here is

$$F(i, j, k) = exp(-\tau[\,(i - i_0)^2 + (j - j_0)^2 + (k - k_0)^2]). \tag{1}$$

3 Algorithm Implementation and Development

3.1 Setup

1. Load `subdata.mat` into MATLAB.
2. Define spacial domain length and Fourier node numbers, and discretize the space into 64*64*64 points to satisfy the 2^n condition.

3. Scale the time domain to $[-2\pi, 2\pi]$ and apply `fftshift()` that swaps the domain.

4. Use the above setup to create a 3-D time space and a 3-D frequency space.

3.2 Averaging of the Spectrum

1. Reshape each column in the dataset to 64*64*64, and apply the Fast Fourier Transform on this 3-D data.

2. Sum up all 49 FFT results, shift them, and take the absolute value and the average of the sum.

3. We can normalize the frequency by dividing the maximum value in the data from the previous step.

4. Locate the maximum value in the averaged frequency and use this location as the center of the filter in the next step. We noticed that the Kx and Ky coordinates need to be switched in order to get the same location as in the visualization plots.

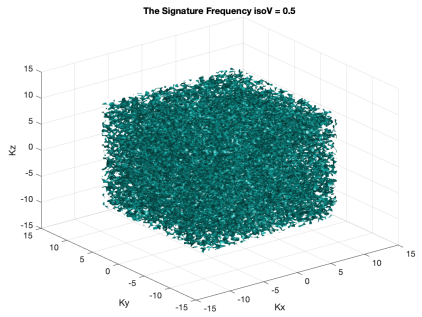

Figure 1: 3-D isosurface plot of the signature frequency with isoV = 0.5

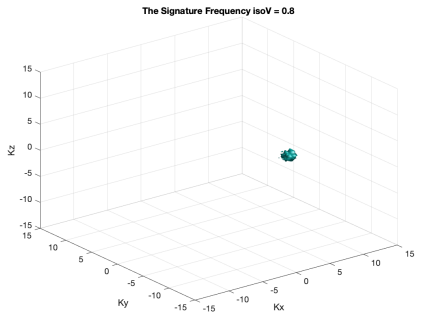

Figure 2: 3-D isosurface plot of the signature frequency with isoV = 0.8

3.3 Applying the Gaussian Filter

1. Set the window width τ and construct a filter around the signature frequency.

2. Apply the filter to each column in the dataset using a for loop. In the loop, we perform FFT on each column and shift it. Then, we multiply the result with the filter function and, lastly, perform the inverse Fourier Transform to return to the space domain.

3. Locate the maximum and the position of the signal, and store it in vectors for each axis accordingly.

3.4 Visualization

The `isosurface()` and `plot3()` functions both are helpful tools for visualizing the path of the submarine, as in Figures 3 - 6.

4 Computational Results

Figure 1 and Figure 2 show the `isosurface()` graphs of the averaged frequency in the frequency domain with isovalues 0.5 and 0.8 respectively. Figure 1 does not provide much useful info but we can clearly see the signature frequency from Figure 2. The coordinates is (5.3407, -6.9115, 1.8850).

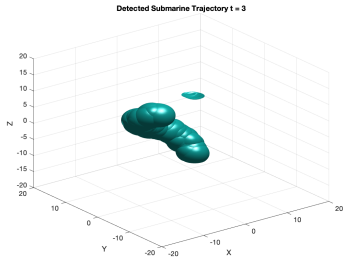

Figure 3: 3-D isosurface of the path with t = 3

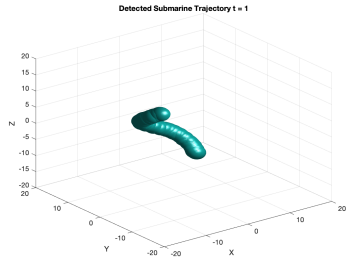

Figure 4: 3-D isosurface of the path with t = 1

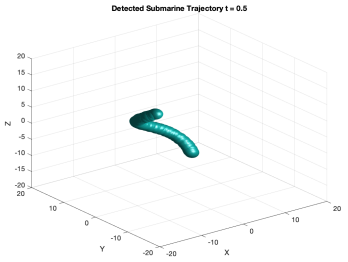

Figure 5: 3-D isosurface of the path with t = 0.5

After having the center, we can apply the 3-D Gaussian filter around its coordinates, using different window sizes for finding the optimal width. Here I plotted the results using multiple τ, from large to small, meaning the width starts small, then becomes larger. As in Figure 3, when $\tau = 3$, a relatively small window, the trajectory is not smooth, and there exist some undesired noise. So I tried $\tau = 1$ and $\tau = 0.5$. The resulting plots, Figure 4 and Figure 5, are very similar, but Figure 5 seems to be smoother. So we will use the trajectory found using filter with $\tau = 0.5$ as the final result. In fact, a range of τ would yield the same result. For example, all numbers in (0.5, 1) would work.

Figure 6 uses plot3 to show the path of the submarine in a 3-D space using line and points. The dot in the red square is the most recent location of the submarine.

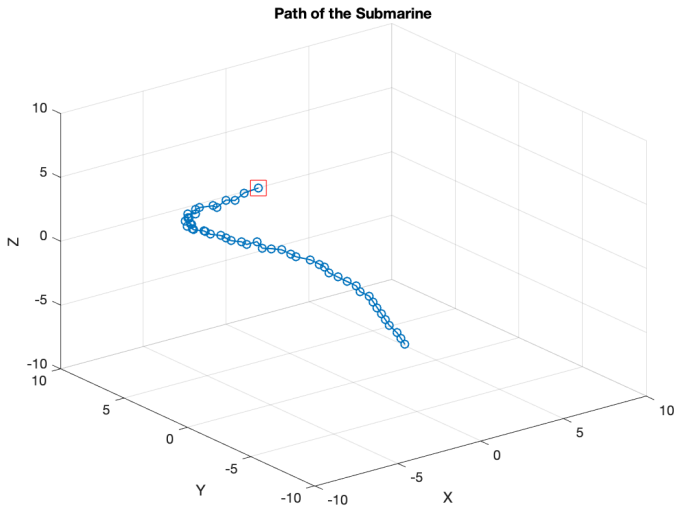

Figure 6: Submarine movements at 49 steps in 3-D space

The coordinates of the 49 time steps are shown in Figure 7 and Figure 8.

	X	Y	Z
1	3.1250	0	-8.1250
2	3.1250	0.3125	-7.8125
3	3.1250	0.6250	-7.5000
4	3.1250	1.2500	-7.1875
5	3.1250	1.5625	-6.8750
6	3.1250	1.8750	-6.5625
7	3.1250	2.1875	-6.2500
8	3.1250	2.5000	-5.9375
9	3.1250	2.8125	-5.6250
10	2.8125	3.1250	-5.3125
11	2.8125	3.4375	-5
12	2.5000	3.7500	-4.6875
13	2.1875	4.0625	-4.3750
14	1.8750	4.3750	-4.0625
15	1.8750	4.6875	-3.7500
16	1.5625	4.6875	-3.4375
17	1.2500	5	-3.1250
18	0.6250	5.3125	-2.8125
19	0.3125	5.3125	-2.5000
20	0	5.6250	-2.1875
21	-0.6250	5.6250	-1.8750
22	-0.9375	5.9375	-1.8750
23	-1.2500	5.9375	-1.2500
24	-1.8750	5.9375	-1.2500
25	-2.1875	5.9375	-0.9375

Figure 7: X, Y, Z coordinates of the submarine in time steps 1 through 25

	X	Y	Z
26	-2.8125	5.9375	-0.6250
27	-3.1250	5.9375	-0.3125
28	-3.4375	5.9375	0
29	-4.0625	5.9375	0.3125
30	-4.3750	5.9375	0.6250
31	-4.6875	5.6250	0.9375
32	-5.3125	5.6250	1.2500
33	-5.6250	5.3125	1.5625
34	-5.9375	5.3125	1.8750
35	-5.9375	5	2.1875
36	-6.2500	4.6875	2.5000
37	-6.5625	4.6875	2.8125
38	-6.5625	4.3750	3.1250
39	-6.8750	4.0625	3.4375
40	-6.8750	4.0625	3.7500
41	-6.8750	3.4375	4.0625
42	-6.8750	3.4375	4.3750
43	-6.8750	3.1250	4.6875
44	-6.5625	2.5000	5
45	-6.5625	2.1875	5
46	-6.2500	1.8750	5.6250
47	-5.9375	1.5625	5.6250
48	-5.6250	1.2500	6.2500
49	-5	0.9375	6.5625

Figure 8: X, Y, Z coordinates of the submarine in time steps 26 through 49

5 Summary and Conclusions

Started with noisy acoustic data emitted by the submarine, we denoised the signals by using Fourier Transform. The signature frequency was found at (5.3407, -6.9115, 1.885) by averaging of the spectrum. Utilizing a Gaussian filter with $\tau = 0.5$, we were able to get the location of the submarine over the past 24-hour period. The last location of the submarine, decoded from the data, is (-5, 0.9375, 6.5625). We have successfully tracked the submarine.

Appendix A MATLAB Functions

- `y = linspace(X1, X2, N)` generates `N` points between `X1` and `X2`, and returns them in a vector.
- `y = fftshift(X)` returns shifted `x` with zero-frequency component at the center of spectrum. For N-dimensional arrays, `fftshift(X)` swaps the first and second half of `X` along each axis.
- `[X, Y, Z] = meshgrid(x, y, z)` returns 3-D grid coordinates with grid vectors `x`, `y` and `z`. The grid represented by `X` and `Y` is of the size `length(y)` * `length(x)` * `length(z)`. So we need to be careful when using the `X` and `Y`.
- `y = reshape(X, M, N, P)` returns a 3-D array with elements in `X` but has the size `M` * `N` * `P`.
- `y = fftn(X)` returns an N-dimensional discrete fast Fourier transform on the N-dimensional array `X`.
- `M = max(X)` returns the largest value in vector `X`. `[M, I] = max(X)` returns the maximum of the N-dimensional array `X` along with its indices.

- `isosurface(X, Y, Z, V, ISOVALUE)` plots the isosurface geometry for data V at isosurface value `ISOVALUE`.
- `[I1, I2, I3] = ind2sub(SIZ, IND)` returns the equivalent indices in the 3-D space of the `IND` passed in with the size specified as `SIZ`.
- `y = ifftn(X)` returns an N-D inverse discrete fast Fourier transform on the N-D array `X`.
- `plot3(X, Y, Z)` plots a line in 3-D space through the points with coordinates `X`, `Y`, and `Z`.

Appendix B MATLAB Code

```
clear all; close all; clc

load subdata.mat
% Imports the data as the 262144x49 (space by time) matrix called subdata

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);
[Kx, Ky, Kz] = meshgrid(ks, ks, ks);

% averaging
avg = zeros(n, n, n);
for j = 1:49
    avg(:, :, :) = avg + abs(fftn(reshape(subdata(:, j), n, n, n)));
end
avg = fftshift(avg)/49;

% figure(1)
isosurface(Kx, Ky, Kz, avg/max((avg(:))), 0.5)
axis([-15 15 -15 15 -15 15])
grid on
drawnow
xlabel("Kx")
ylabel ("Ky")
zlabel ("Kz")
title ("The Signature Frequency isoV = 0.5")
saveas(gcf, 'Signature0.5.png')
clf

isosurface(Kx, Ky, Kz, avg/max((avg(:))), 0.8)
axis([-15 15 -15 15 -15 15])
grid on
```

```
drawnow
xlabel("Kx")
ylabel ("Ky")
zlabel ("Kz")
title ("The Signature Frequency isoV = 0.8")
saveas(gcf, 'Signature0.8.png')
clf

[maximum, index] = max(avg(:));
[i1, i2, i3] = ind2sub([n, n, n], index);
xx = ks(i2);
yy = ks(i1);
zz = ks(i3);

% Gaussian filtering
% figure(2)
clf
a = 3;
filter = exp(-a * ((Kx - xx).^2 + (Ky - yy).^2 + (Kz - zz).^2)) ;
locX = zeros(1,49);
locY = zeros(1,49);
locZ = zeros(1,49);
for j = 1:49
    Un(:, :, :) = reshape(subdata(:, j), n, n, n);
    utn = fftn(Un);
    unft = filter .* fftshift(utn);
    unf = ifftn(unft);
    [M, I] = max(unf(:));
    [i1, i2, i3] = ind2sub(size(unf), I);
    locX(j) = x(i1);
    locY(j) = y(i2);
    locZ(j) = z(i3);
    isosurface(X, Y, Z, abs(unf) / max(abs(unf(:))), 0.5)
    axis([-20 20 -20 20 -20 20])
    grid on
    drawnow
    xlabel("X")
    ylabel ("Y")
    zlabel ("Z")
    title ("Detected Submarine Trajectory t = 3")
end
saveas(gcf, 'Trajectory3.png')
clf

a = 1;
filter = exp(-a * ((Kx - xx).^2 + (Ky - yy).^2 + (Kz - zz).^2)) ;
locX = zeros(1,49);
locY = zeros(1,49);
locZ = zeros(1,49);
for j = 1:49
    Un(:, :, :) = reshape(subdata(:, j), n, n, n);
```

```
    utn = fftn(Un);
    unft = filter .* fftshift(utn);
    unf = ifftn(unft);
    [M, I] = max(unf(:));
    [i1, i2, i3] = ind2sub(size(unf), I);
    locX(j) = x(i1);
    locY(j) = y(i2);
    locZ(j) = z(i3);
    isosurface(X, Y, Z, abs(unf) / max(abs(unf(:))), 0.5)
    axis([-20 20 -20 20 -20 20])
    grid on
    drawnow
    xlabel("X")
    ylabel ("Y")
    zlabel ("Z")
    title ("Detected Submarine Trajectory t = 1")
end
saveas(gcf, 'Trajectory1.png')
clf

a = 0.5;
filter = exp(-a * ((Kx - xx).^2 + (Ky - yy).^2 + (Kz - zz).^2)) ;
locX = zeros(1,49);
locY = zeros(1,49);
locZ = zeros(1,49);
for j = 1:49
    Un(:, :, :) = reshape(subdata(:, j), n, n, n);
    utn = fftn(Un);
    unft = filter .* fftshift(utn);
    unf = ifftn(unft);
    [M, I] = max(unf(:));
    [i1, i2, i3] = ind2sub(size(unf), I);
    locX(j) = x(i2);
    locY(j) = y(i1);
    locZ(j) = z(i3);
    isosurface(X, Y, Z, abs(unf) / max(abs(unf(:))), 0.5)
    axis([-20 20 -20 20 -20 20])
    grid on
    drawnow
    xlabel("X")
    ylabel ("Y")
    zlabel ("Z")
    title ("Detected Submarine Trajectory t = 0.5")
end
saveas(gcf, 'Trajectory0.5.png')
clf

% figure(3)
plot3(locX, locY, locZ, '-o', 'LineWidth', 1)
axis ([ -10 10 -10 10 -10 10])
xlabel("X")
```

```
ylabel ("Y")
zlabel ("Z")
grid on
hold on
plot3(locX(49), locY(49), locZ(49), 'rs', 'MarkerSize', 15)
title ("Path of the Submarine")
saveas(gcf, 'plot3.png')
clf

% print the coordinates
% figure(4)
coord = {'X';'Y';'Z'};
T = table(locX', locY', locZ');
uitable('Data', T{1:25,:}, 'ColumnName', coord, 'RowName', ...
    1:25 ,'Units', 'Normalized', 'Position', [0, 0, 1, 1]);
saveas(gcf, 'table1.png')

uitable('Data', T{26:49,:}, 'ColumnName', coord, 'RowName', ...
    26:49 ,'Units', 'Normalized', 'Position', [0, 0, 1, 1]);
saveas(gcf, 'table2.png')
```