AMATH 482 HW1 - A Submarine Problem

Winnie Shao

1/27/2021

Abstract

This report discuss a method using Fourier transform to reduce the noise in the acoustic data. Our goal is to hunt for a submarine in the Puget Sound using 49 measurements in some spatial domain. To obtain the solution, we will implement the fast Fourier transform and its inverse to switch the data between the spatial domain and frequency domain. More specifically speaking, we are going to find the frequency signature and extract intuitive information from the spectrum.

Section I: Introduction and Overview

In this problem, we are hunting for a submarine in the Puget Sound using noisy acoustic data. It is a new submarine technology that emits an unknown acoustic frequency that we need to detect. We need to find the track of the submarine and send our aircraft to it. To do that, we to use a broad spectrum recording of acoustics data over a 24-hour period in half-hour increments and apply Fourier Transform on the data to obtain the final position of the submarine.

Fourier transform is considered one of the most efficient methods for solving problems related to time-frequency analysis. It is widely used in signal processing. For solving our problem, we are going to apply the Fast Fourier Transform which is an efficient and accurate algorithm that performs forward and backward Fourier transform so that we can extract the important information out of the noisy data set.

In the following discussion, I will be presenting the theoretical concept about FFT along with other techniques such as averaging and filtering which are also essential to solve this problem. I will also walk through my implementation in MATLAB to make the code in Appendix B more accessible.

Section II: Theoretical Background

2.1 Fourier Transform

Fourier transform is a tool to decompose a function from time domain to frequency domain. It is an integral defined over the entire line $x \in [-\infty, \infty]$ and computed over a finite domain $x \in [-L, L]$. This tool can be used to transform a sophisticated function into another form and decompose it into a frequency-amplitude image.

When $x \in [-\infty, \infty]$, the Fourier transform in one dimension is defined as:

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

The inverse Fourier transform is defined as:

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

2.2 The Fast Fourier Transform

To solve this problem, we will apply the Fast Fourier Transform (FFT algorithm) to perform the Fourier transform and inverse Fourier transform. It is widely used because of its low operation count O(NlogN) and good accuracy. In MATLAB, we use functions such as fft(x), ifft(x), fftshift(x), ifftshift(x), and fftn(x) to perform forward, backward, one-dimensional and multidimensional FFT. One thing that we need to pay attention on is that the algorithm associated with the FFT assumes we are working on a 2π periodic domain so we are going to rescale the frequency by $\frac{2\pi}{L}$ in our implementation.

2.3 Gaussian Filtering

Although it seems really practical to implement FFT, most data in real world situation will have noises mixed with the signals we receive. To prevent the error from the noise, we need to perform frequency filtering before apply FFT. Spectral filtering is a method allows us to extract information at specific frequency. It can removing most of the white-noise and filter out our desired frequency. The filter we used in this problem is the Gaussian filter:

$$F(k) = exp(-\tau(k - k_0)^2) \tag{3}$$

Where τ is the bandwidth of the filter and k_0 is the center frequency of the desired signal field.

2.4 Averaging

Averaging is a method we use to extract the clean spectral signature, or central frequency. Central frequency is a specific frequency or frequency range of the target object from all the noisy signals. The main idea of this method is that white-noise over signals should, on average, add up to zero. Therefore, by averaging a given set of realization, we can remove most of the white-noise and get a distinguishable structure of central frequency. The only downside of this method is that in the averaging process, we lose the time domain dynamics.

Section III: Algorithm Implementation & Development

3.1 Initialization

First of all, we load data from subdata.mat, which gives the 49 measurements about the submarine. Then, we define the spatial domain and frequency domain (number of frequencies in each direction in the spatial domain). We divide x, y, z $\in [-L, L]$ in spatial domain into n discrete modes. Then, we rescale the frequency domain by multiplying them with $\frac{2\pi}{L}$ because the FFT assumes a 2π periodic domain. After this, we also shift the aces of the frequency domain. At last, we use meshgrid to make 3-D arrays which are X-Y-Z coordinate system and Kx-Ky-Kz frequency system.

3.2 Averaging

In this part, we first reshape to fit the relization data into three dimension and use fftn() to perform multi-dimensional FFT to add up the frequency. Once we get the accumulated frequencies of signals, we rescale it. Since it might have complex numbers, we take the absolute value of the accumulate frequencies to get the intensity of the signal. Then we search for the strongest signal in the array and divide the array bu the strongest signal. Then we use find() to look for the coordinates of the strongest signal.

3.3 Filtering

For the filtering part, we create a multi-dimensional Gaussian filter using the equation (3). We did the same thing in the averaging section, reshaping the data for each realization. Then, we apply the filter by multiplying the transformed data in each direction. We take absolute value of the result and remove the effects of complex number and then search for the signal with max absolute value. Based on the coordinates of the max signal for each realization, we can locate the position of the submarine. In the end, we save the position information into an array.

3.4 Output

Based on the position information we obtained from all the steps above, we can plot out the path of the submarine by using plot3(). The position we should send our sub-tracking aircraft should be the coordinates stored in the last row of the position array.

Section IV: Computational Results

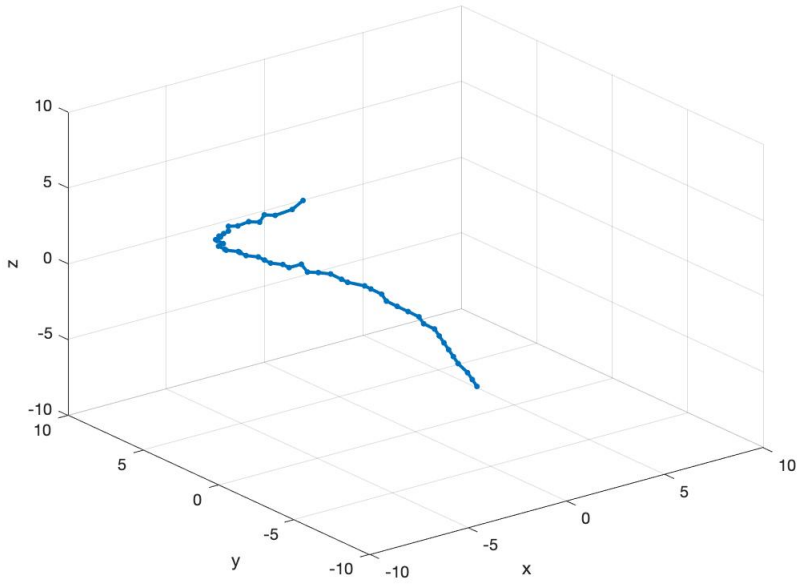
Through averaging of the spectrum, we can determine the central frequency of the submarine which is

$$[kx, ky, kz] = [5.4307, -6.9115, 2.1991] \tag{4}$$

By applying the Gaussian filter, we denoise the 49 measurements and plot the path of the submarine in Figure 1.

Finally, we should send our aircraft to x-y position: $[x, y] = [-5, 0.9375]$

Figure 1: The Path of Submarine



Section V: Summary and Conclusions

By implementing the procedure of averaging, we remove the white-noise and obtain the frequency signature. Then, after applying the Gaussian filter with frequency signature in each direction, we denoise the dataset in the spatial domain and track the submarine. This success illustrated that the method we implemented with FFT is helpful in solving problems like this. In this end, we got the final position which is (-5, 0.9375) so now we can send our tracking aircraft to follow it!

Appendix A: MATLAB functions used and brief implementation explanation

Function	Syntax	Brief Explanation
find	[row, col] = find()	Returns the row and column of each non zero element in an array with given input arguments
fftn	fftn(X)	Returns the multidimensional Fourier transform of an N-Dimension array X using FFT
fftshift	fftshift(X)	Rearranges a Fourier transform X by shifting the zero frequency component to the center.
ifftn	ifftn(Y)	Returns the multidimensional discrete inverse Fourier transform of an N-D array using FFT
ifftshift	ifftshift(Y)	Rearranges a shifted Fourier transform Y back to the original transform output. The inverse of fftshift(X)
isempty	TF = isempty(A)	Rturns logical 1 if A is empty and logical 0 otherwise
isosurface	isasurface(X,Y,Z,V, isovalue)	Computes isosurface data from the volume data V at the isosurface value specified in isovalue, and creates 3-D visualizations
meshgrid	[X,Y,Z] = meshgrid(x, y, z)	Returns 3-D grid coordinates defined by the vector x, y, z
plot3	plot3(X,Y,Z, lineSpec)	Creates the plot using the specified line style, marker, and color
reshape	B = reshape(A, sz1, ..., szN)	Reshape a multidimensinal array into a matrix with specific size

Appendix B: MATLAB codes

```
clear; close all; clc;
load subdata.mat

% Initialization
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);

% Part 1
ave = zeros(n, n, n);
for j=1:49
    Sn(:,:,:)=reshape(subdata(:, j),n,n,n);
    Stn = fftn(Sn);
    ave = ave + Stn;
end

ave = abs(fftshift(ave));
flatAve = reshape(ave, n^3, 1);
strongest = max(flatAve);
ave = ave/strongest;

ave = ifftshift(ave);
```

```
for m = 1:n
    [j, i] = find(ave(:,:,m) == 1);
    if isempty(i)~=1
        center_frequency = [i, j, m];
        break
    end
end
kx=k(center_frequency(1));
ky=k(center_frequency(2));
kz=k(center_frequency(3));

% Part 2: filter
tau = 1.0;
filter = exp(-tau*(fftshift(Kx)-kx).^2).*...
    exp(-tau*(fftshift(Ky)-ky).^2).*...
    exp(-tau*(fftshift(Kz)-kz).^2);

position = zeros(49, 3);

for realize = 1:49
    % apply filter on each realization
    Sn(:,:,:)=reshape(subdata(:, realize),n,n,n);
    Stn = fftn(Sn);
    snft = filter.* Stn;
    snf = ifftn(snft); % inverse multi-dimensional FFT
    snf = abs(snf);
    flat_snf = reshape(snf, n^3, 1);
    strongest = max(flat_snf);
    % look for the strongest signal
    for m = 1:n
```

```
            [j, i] = find(snf(:,:,m) == strongest);
            if isempty(i)~=1
                position(realize,:) = [x(i), y(j), z(m)];
                break
            end
        end

        isosurface(X,Y,Z,snf/strongest,0.8)
        axis([-L L -L L -L L]), grid on, drawnow
        xlabel('x'); ylabel('y');zlabel('z');
        pause(1)
end

plot3(position(:,1), position(:,2), position(:,3), '.-', 'MarkerSize',...
    10, 'Linewidth', 2)
axis([-L L -L L -L L]), grid on, drawnow
xlabel('x'); ylabel('y'); zlabel('z');

[kx, ky, kz]
solution = [position(49,1), position(49,2),position(49,3)];
```