AMATH 482 Homework 1

Rui Zhu

January 18, 2021

Abstract

This report aims to de-noise and track the path of a submarine from a set of noisy acoustic data.

1 Introduction and Overview

Fourier Transform (FT) takes a function of space (or time), x, and converts it to a function of frequencies, k. However, Fourier Transform assumes infinite domain so is not necessarily what we want for the this practical situations. In this assignment, we will use Fast Fourier Transform (FFT) instead.

We are given a raw dataset contains 49 time-measurements of noisy acoustic data. And our goal is to de-noise this data and to locate the submarine. To do so, we first determine the frequency signature (center frequency) generated by the submarine using the method of averaging. Then we apply a typical Gaussian filter to the data around the center frequency in order to denoise the data and determine the path of the submarine. Finally, we report the x and y coordinates in a table to send our P-8 Poseidon subtracking aircraft.

This assignment consists of seven sections (click on it to go directly):

2 Theoretical Background

2.1 The Fourier Transform

As we learned from our textbook [1], the Fourier Transform takes a function of space (or time), x, and converts it to a function of frequencies, k. It is defined as

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

And Inverse Fourier Transform is defined as

$$f(x) = \frac{2}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(k) e^{-kx}\, dk \tag{2}$$

2.2 The Fast Fourier Transform

The FFT is an algorithm that does the Discrete Fourier Transform faster ($\mathcal{O}(N \log N)$). One important thing to notice is that the FFT algorithm requires the domain to be discretized by 2^n, and the boundary conditions to be periodic. As we learned in class, the FFT algorithm works extremely well compared to finite difference.

2.3 Filtering and Averaging

A common method to locate the center frequency is Gaussian function, it is defined as

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

where τ determines the width of the filter and the constant k_0 determines the centre of the filter. Notice that in this assignment we use a Gaussian function in 3-d, and this is similarly defined as

$$F(k) = e^{-\tau[(Kx-ki)^2+(Ky-kj)^2+(Kz-kk)^2]} \tag{4}$$

A key for assuming *white noise* is *zero mean*. That is, given a bunch of signals whose frequency is unknown but constant, if we average over many realizations (in frequency space), the noise from each realization will cancel out and we will get something close to the true signal. Once we find the frequency centre using time averaging, our Gaussian filter can be applied around the centre then applied to all signals used in the process of averaging.

3 Algorithm Implementation and Development

3.1 Initial Setup

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

2. Define our domain and grid vectors.

   (a) Notice that we need to **rescale frequencies** for the FFT algorithm since it assumes 2π periodic signals.

   (b) We also need to use `fftshift` to shift the transformed signial back to the 'normal' order.

3.2 Part 1: Averaging The Spectrum & Determine the Center Frequency

1. Define the average array.

2. Averaging The Spectrum & Locate the centre.

Algorithm 1:

Data: `subdata.mat`

for $j = 1 : 49$ **do**
  Extract noisy measurement from `subdata` at each timestep j
  Transform the current noisy measurement into frequency domain using `fftn` `// notice the command choice for Higher Dimensional Fourier Transforms here`
  Add the transformed signal to the average array
Divide the average by 49
Find the center frequency (place of largest magnitude) using `max(), abs(), and ind2sub()`.
`// Notice the use of ind2sub() to switch between 1D and 3D for convenience`

3. Plot the centre frequency using `isosurface()`. `// Notice we normalize this 3D matrix using the largest element in it`

3.3 Part 2: Filtering & Determine the Path

1. Define a 3*49 coordinate array

2. Define 3D Gaussian function as filter.

3. Apply this filter to each frequency matrix then use `ifftn` to transform back to space domain.

4. Use plot3 to plot the path of the submarine, each row of the coordinate array represents the x, y, and z coordinates, respectively.

Algorithm 2:

Data: `subdata.mat`

for $j = 1 : 49$ **do**

- Reshape the 1D data at each time step j to 3D
- Use `fftn` to transform to frequency domain `// Again notice the use of fftshift`
- Apply our Gaussian Filter `// simply multiply them together`
- Use `ifftn` to transform back to space domain
- Store coordinates of submarine at each time using `max(), abs(), ind2sub()` `// Same as before`
- Save the x,y,z coordinate at this timestep into our coordinate array

3.4 Part 3: P-8 Poseidon subtracking aircraft

Record the x and y coordinate at the last timestep in our coordinate array to send the aircraft.

4 Computational Results

Figure 1 shows the frequency centre of the signals after time averaging

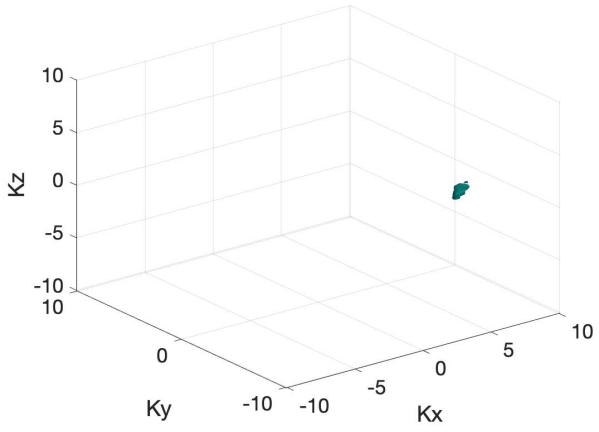

Figure 1: Frequency Centre

Figure 2 shows the path of the submarine

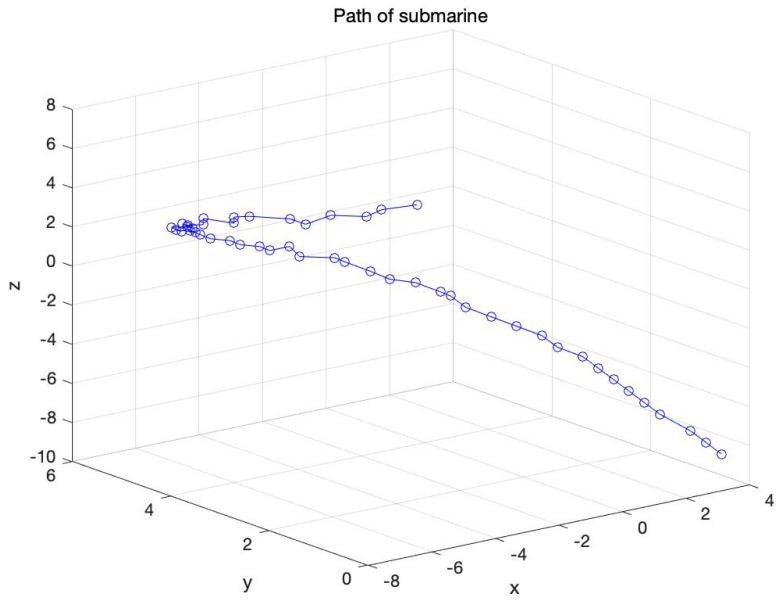

Figure 2: Path of Submarine

Table 1 shows the final location of the submarine

Axis	Coordinate
X	-5
Y	0.9375

Table 1: Final location of the submarine

5 Summary and Conclusions

Filtering by Gaussian function and time averaging were successfully applied to de-noise our raw acoustic data. We obtained the frequency centre by averaging the 49 signals, and we applied our 3D Gaussian filter based on that center. Finally, we successfully record the path of the submarine and the final location of it.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.

- `[X,Y] = meshgrid(x,y)` returns 2-D grid coordinates based on the coordinates contained in the vectors `x` and `y`.
- X is a matrix where each row is a copy of `x`, and
- `Y` is a matrix where each column is a copy of `y`. The grid represented by the coordinates `X` and `Y` has
- `length(y)` rows and `length(x)` columns.
- `[X,Y,Z]=ind2sub([n n n] index)` returns a 1*3 array based on the input dimensions and a 1D index.
- `fftn()` decomposes the signal into frequencies in each direction.
- `ifftn()` decomposes the signal into time/space in each direction.
- `fftshift()` the transformed ut has an order of $k = 0, 1, ..., \frac{N}{2} - 1, -\frac{N}{2}, ..., -1$ this function shift it back to match the ut with frequency
- `plot3(X,Y,Z,style)` plots coordinates in 3-D space.
- `grid on` displays the grid in plots
- `isosurface(X,Y,Z,volume,isovalue,style)` computes isosurface data from the volume data at the isosurface value specified in isovalue.
- `array2table()` takes an array and display it as a table

Appendix B MATLAB Code

```
%% Setup

% Clean workspace
clear all; close all; clc

load('subdata.mat') % Imports the data as the 262144x49 (space by time) matrix called subdata 5

%Define our domain
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: Averaging The Spectrum & Determine the Center Frequency

ave = zeros(n,n,n); %defining the average array
for j=1:49
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    Unt = fftn(Un);
    ave = Unt+ave;
end
ave = abs(fftshift(ave))/49;

% Find the center frequency (place of largest magnitude)
[M, linearInd] = max(abs(ave(:)));
[I,J,K] = ind2sub([n n n], linearInd);

% Get frequency components for the center frequency
ki = Kx(I,J,K);
kj = Ky(I,J,K);
kk = Kz(I,J,K);

figure(1)

% Plotting the center frequency
isosurface(Kx,Ky,Kz,abs(ave)/max(ave(:)),0.6, 'r');
set(gca, 'FontSize',18);
```

```
axis([ks(1) -ks(1) ks(1) -ks(1) ks(1) -ks(1)]), grid on;
xlabel('Kx'); ylabel('Ky'); zlabel('Kz');

%% Part 2: Filtering & Determine the Path

% Assuming white noise, using Gaussian as fliter
filter = exp(-((Kx - ki).^2 + (Ky - kj).^2 + (Kz - kk).^2));



% Use this filter to denoise and locate the submarine
% Apply filter to denoise each frequency matrix then take ifft
sub = zeros(3,49);
for j=1:49
    Un = reshape(subdata(:,j),n,n,n);
    Unt = fftshift(fftn(Un));


    Unft = Unt .* filter;
    Unf = ifftn(Unft);

    % Store coordinates of submarine at each time
    [~, ind] = max(abs(Unf(:)));
    [subi, subj, subk] = ind2sub([n n n], ind);
    sub(1,j) = X(subi, subj, subk);
    sub(2,j) = Y(subi, subj, subk);
    sub(3,j) = Z(subi, subj, subk);
end

%Plot the path
figure(1)
plot3(sub(1,:), sub(2,:), sub(3,:),'b-o'), grid on;
title('Path of submarine'), xlabel('x'), ylabel('y'), zlabel('z')

% %Path of the submarine without noise
```

```
% for j=1:49
%     M = max(abs(Unf),[],'all');
%     close all, isosurface(X,Y,Z,abs(Unf)/M,0.7)
%     axis([-20 20 -20 20 -20 20]), grid on, drawnow
%     pause(1)
% end

% %Path of the submarine with noise
% for j=1:49
%     Un(:,:,:)=reshape(subdata(:,j),n,n,n);
%     M = max(abs(Un),[],'all');
%     close all, isosurface(X,Y,Z,abs(Un)/M,0.7)
%     axis([-20 20 -20 20 -20 20]), grid on, drawnow
%     pause(1)
% end

%% Part 3: P-8 Poseidon subtracking aircraft

time = [sub(1,:); sub(2,:)];
array2table(time)
```