AMATH482 Homework 1:A Submarine Problem

Haoran Sun

January 18, 2021

Abstract

An object can be tracked by analyzing its emitted signals. However, the signals always come with noise in practice and therefore, denoising is necessary . This text introduces a method to denoise the acoustic data from a submarine. The method determines the central frequency by averaging the data in the frequency domain and applies a spectral filter centered on the determined frequency to filter out the noise. A result of the precise path of the submarine is obtained using the filtered data.

1 Introduction and Overview

This text describes an application of denoising in a real situation: Submarine Tracking. A set of acoustic signals emitted from a moving submarine is obtained over a 24-hour period in half-hour increments. Using the data, we can locate the submarine at each time the data is obtained, and find its trajectory. However, the data, as a situation in real life should always be, is noisy. It is necessary to denoise the data in order to track the path of the submarine precisely.

1.1 Averaging

The original data is in space domain. Since the noise is affecting the signal in frequency domain, we will first take the Fourier Transform of the signal, presenting it in its signal domain.We have the assumption that this noise is white noise, which means it affects all the frequency the same. The noise has a mean of zero. Hence, if we have multiple realizations of the signal and take the average of them in frequency space, the noise will cancel out and the frequency signature(central frequency) of the signal will be highlighted.

1.2 Filtering

Once the central frequency is known, we will be applying a filter,which is able to attenuate undesired frequencies and noise while maintain the central frequency. A filter is just a function our signal will be multiplied by. In this text, we will simply use Gaussian function as our filter. After the signal is filtered, we transfer it back to space domain from frequency domain using Inverse Fourier Transform. The signal becomes clean and can be used to determine submarine's location and path accurately.

2 Theoretical Background

2.1 Fourier Transform

2.1.1 Fourier Transform and Inverse Fourier Transform

From our textbook [1], we define the Fourier Transform of a given function $f(x)$, where $x \in \mathbb{R}$, written as $\hat{f}(k)$, by the formula:

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

This equation decomposes the given function, which is in spatial or time domain, into function depending on frequency. The function $\hat{f}(k)$ in frequency domain can be also recovered back to $f(x)$ in spatial or time domain using Inverse Fourier Transform, defined as:

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

2.1.2 Discrete Fourier Transform and Fast Fourier Transform

Discrete Fourier Transform(DFT) has the same use as Fourier Transform, representing a function in its frequency domain. However, instead of integrating over infinite domain, DFT takes N values $\{x_0, x_1, ...x_{n-1}\}$ sampled at equally-space points from a function, and outputs a sequence of numbers given by:

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}} \tag{3}$$

In the equation, only finite frequencies present: $k = 0, 1, 2...N - 1$. In real-life situation, DFT is more practicable than Fourier Transform, since few signals can be obtained on infinite domains.

Fast Fourier Transform(FFT) is an algorithm that does DFT in a faster way. For DFT described above, there is a total complexity of $\mathcal{O}(N^2)$ for N points, which can be very large when N is large. The basic logistic of FFT is, for a sequence of length N, the DFT will be split into two DFTs, each of sequence of length N/2. Then the algorithm repeats this step over and over. FFT is performed with the total complexity of $\mathcal{O}(Nlog(N))$, reducing the order of operations by a large amount.

2.2 Gaussian Filter

A Gaussian Function is defined as:

$$F(k) = e^{-\tau(k-k_0)^2} \tag{4}$$

We will multiply the signal of the submarine by this function as our filter, where the parameter τ determines the width of the filter, and the constant k_0 determines the center of the filter. We will use the value of the central frequency as k_0 so that the filter will be centered at the central frequency and maintain its value.

3 Algorithm Implementation and Development

The signal are provided as a 262144*49, space by time matrix. It contains 49 columns of data for measurements over a 24-hour span in half-hour increments. We set L = 10 as our spatial domain and We will sample the function at equally-spaced 64 points.We set up the frequency domain and re-scale it by $2\pi/L$, since the FFT assumes 2π periodic signals.

3.1 Average the signal

For averaging the signal, we first add up all the realizations of the signal, and then take its average. In our algorithm(see Algorithm 1), we will:

1. Create a zero 3-d matrix with the dimension of the reshaped signal, as our sum variable.
2. Create a loop to go over the data at each time.
3. Inside the loop, reshape the data of the corresponding time
4. Transfer the signal into its frequency domain and add this signal to the current sum variable
5. After the loop, we divide the sum variable by 49, the number of realizations we added.
6. Draw the isosurface of the averaged signal in 3D coordinates, which is defined previously and obtain the value of the central frequency.

Algorithm 1: Averaging the Data

```
Import data from subdata.mat
Build a zero n*n*n matrix named sum
for $j = 1 : 49$ do
    Reshape the data on j columns into n*n*n matrix
    Take the FFT of the reshaped data using fftn()
    Add the transferred data into sum
end for
Average the spectrum by dividing sum by 49
Draw the Isosurface of the averaged spectrum using isosurface
```

3.2 Filter and Plot the signal

Once we obtain the value of the central frequency on each dimension, we can define our Gaussian filter,which is in the form of eq.(4). Then the filter will be applied on the signal by simply multiplying the filter and the signal in frequency domain. After filtering, we use the Inverse Fourier Transform to recover the signal back to its spatial domain, which is denoised. In our algorithm(see Algorithm 2), we use 0.2 as the value of τ. This development will be performed as:

1. Set up the Gaussian Filter on each dimension and multiply them together to obtain an overall filter, centered at the central frequencies in their corresponding dimensions.
2. Create a loop to go over the data at each time.
3. Inside the loop, reshape the data of the corresponding time(same as in Algorithm 1)
4. Transfer the signal into its frequency domain and multiply it with the defined filter
5. Transfer the filtered signal back to its spatial domain, and record the coordinates information at each moment into a matrix.
6. Using the matrix that contains the coordinates information, we plot the path and the final location of submarine.

Algorithm 2: Filtering and Plotting

```
Create a 49*3 matrix coord to record coordinates at each time
Define the filter
for $k = 1 : 49$ do
    Reshape the data from subdata.mat on k columns into n*n*n matrix Un.
    Take the FFT of Un using fftn().
    Use fftshift() to correct the order of the frequency domain and multiply the data in frequency
    domain with filter.
    Use ifftshift() to restore the default order of the frequency domain, and do Inverse Fourier
    Transform of the filtered data using ifftn().
    Using max() and ind2sub to extract the coordinates in the filtered data with maximum value.
end for
Using plot3 to plot the path and final position of the submarine in 3d coordinate
```

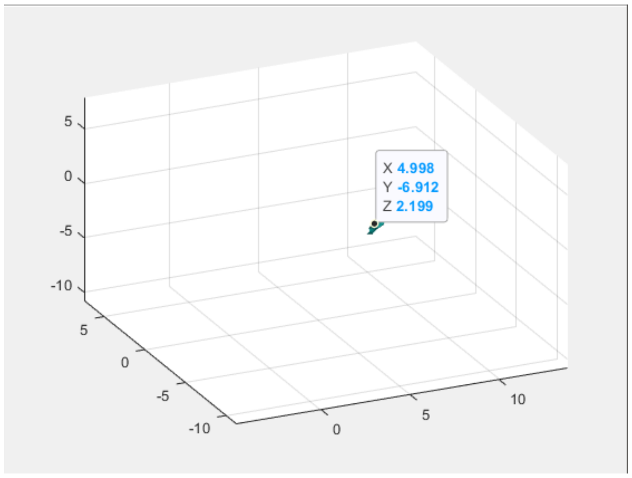

Figure 1: Isosurface of the averaged signal in 3d frequency domain

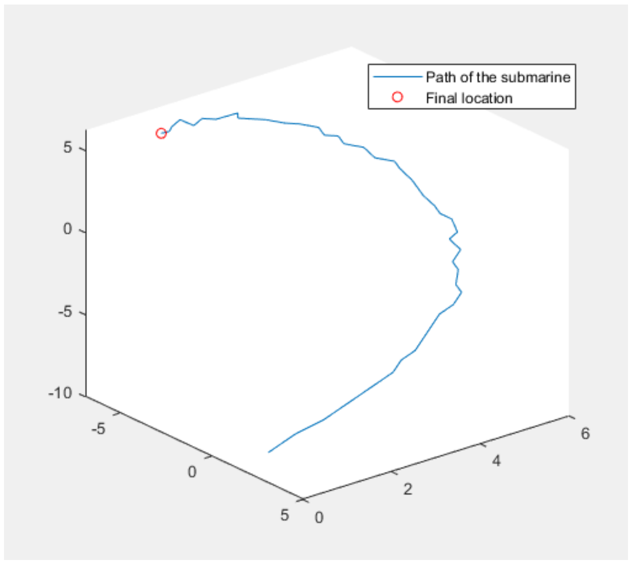

Figure 2: Path and Final Location of Submarine

4 Computational Results

After computing and drawing the isosurface of the averaged signal in frequency domain, we obtain Figure 1. From the figure, it is clear all the data gathered around the coordinates (5,-7,2) approximately. Therefore, we will set the value of the central frequencies to be $k0_x = 5, k0_y = -7, k0_z = 2$. These are also where the Gaussian Filter will be centered at.
After executing Algorithm 2 described previously, the path and the final location of the submarine are shown in Figure 2.
The x and y coordinates of the final location of the submarine is [-5.0000, 0.9375].

5 Summary and Conclusions

This text introduces a method to eliminate the white noise exerted on the signal in the frequency domain. In summary, the method follows the step:

1. Transfer each of the input spatial-dependent realization emitted from the submarine into its frequency domain using FFT.
2. Calculate the average of multiple(49 in this context) realizations and white noise will be attenuated since it has a mean of 0. Desired frequency signatures will be maintained.
3. Define a Gaussian Filter centered at the frequency signatures, and multiply the filter with each realizations in frequency domain. The noise of each realizations will be filtered out.
4. Transfer the filtered signal back to its spatial domain using IFFT.

Since the signal is clean after filtering, accurate coordinates of path and location information can be extracted and visualized. The overall result we achieved meets expectation decently.However, One thing to notice is that the plotted path in Figure 2 is not completely smooth, which can be improved potentially by designing a more developed algorithm.

References

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

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `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.
- `reshape(X,M,N,P)` returns an N-D array with the same elements as `X` but reshaped to have the size M-by-N-by-P.
- `M = max(X,[],'all')` returns the largest element of `X`.
- `isosurface(X,Y,Z,V,ISOVALUE)` computes isosurface geometry for data `V` at isosurface value ISOVALUE. Arrays (X,Y,Z) specify the points at which the data V is given.
- `fftn(X)` returns the N-dimensional discrete Fourier transform of the N-D array X. If X is a vector, the output will have the same orientation.
- `fftshift` shifts zero-frequency component to center of spectrum.
- `ifftn(F)` returns the N-dimensional inverse discrete Fourier transform of the N-D array F. If F is a vector, the result will have the same orientation.
- `plot3(x,y,z)`, where x, y and z are three vectors of the same length, plots a line in 3-space through the points whose coordinates are the elements of x, y and z.

Appendix B MATLAB Code

```
% Clean workspace
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);

%Plot the original noised signal
ave = zeros(n,n,n);
for j=1:49
Un(:,:,:)=reshape(subdata(:,j),n,n,n);
M = max(abs(Un),[],'all');
Unt = fftn(Un);
ave = ave+Unt;
close all;
isosurface(X,Y,Z,abs(Un)/M,0.7)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
pause(0.1);
end
%%
%Averaging the spectrum
aved = abs(fftshift(ave))/49;
M = max(abs(aved),[],'all');
close all
isosurface(Kx,Ky,Kz,abs(aved)/M,0.7)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
K0_x = 5;
K0_y = -7;
K0_z = 2;
tau = 0.2;
```

Listing 1: Code for loading and averaging the signal

```
%%
%Define the filter
filter_x = exp(-tau*(Kx - K0_x).^2);
filter_y = exp(-tau*(Ky - K0_y).^2);
filter_z = exp(-tau*(Kz - K0_z).^2);
filter = filter_x.*filter_y.*filter_z;

%%
%Find the path
close all;
coord = zeros(49,3); %giving the coordinates of positions at each time
for k = 1:49
    Un(:,:,:)=reshape(subdata(:,k),n,n,n);
    Unt = fftn(Un);
    Untf = fftshift(Unt).*filter;
    Untf = ifftshift(Untf);
    Unf = ifftn(Untf);
    Unf = Unf/max(abs(Unf(:)));

    [M_Unf,I] = max(abs(Unf(:)));
    [x_coord,y_coord,z_coord] = ind2sub(size(Unf),I);
    coord(k,1) = X(x_coord,y_coord,z_coord);
    coord(k,2) = Y(x_coord,y_coord,z_coord);
    coord(k,3) = Z(x_coord,y_coord,z_coord);
end
%%
%Plot the path
figure(3)
plot3(coord(:,1),coord(:,2),coord(:,3));
hold on;
plot3(coord(49,1),coord(49,2),coord(49,3),'ro');%final position
legend('Path of the submarine','Final location');

%%
%Table of final x&y coordinates of the submarine
T = [coord(49,1),coord(49,2),coord(49,3)]
```

Listing 2: Code for filtering and plotting