Signal Denoising and Submarine Tracking

Avery Milandin

January 27, 2020

Abstract

In this project we use noisy data gathered in half hour increments to track the path of a submarine as it moves through the ocean over the course of two days. We do this in order to determine where a a P-8 Poseidon subtracking aircraft should fly in order to stay directly above the submarine, and we present this data as a table of x-y coordinates.

1 Introduction and Overview

Our data is given to us in arrays of 262,144 complex numbers, which can be reshaped into a 64x64x64 matrix where each data point represents an amplitude of an acoustic signature at that point in 3-D space. We have 49 such arrays that were recorded at half hour intervals. The task that we are faced with is denoising the data to determine the sound frequency produced by the submarine, and once we know this frequency, we need to track the change in the location from which this frequency is emitted over time and hence track the submarine. Once we know the path of the submarine, we can extract the x and y coordinate of each point in the path, which tells us where the subtracking aircraft should be at each 30 minute increment.

2 Theoretical Background

As we learned in class, if we want a way to visualize the individual frequencies that make up a signal we can apply a Fourier Transform to the signal, $f(t)$, to obtain a transformed signal, $\hat{f}(t)$:

$$\hat{f}(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-ikt} dt$$

If we plot this function, we will have frequency on the x-axis instead of time, and on the y-axis we will have the amplitude of those frequencies, so a Fourier Transform essentially allows us to see how much of each frequency is in our signal. If we would like to apply this to our data, however, we won't be able to because the Fourier Transform assumes that $f(x)$ is a continuous integrable function, but in our case we have only discrete data points sampled at different points in time. This is where the Discrete Fourier Transform comes in. If we have n data points in an array, $[x_0, x_1, x_2 ... x_{n-1}]$, then the Discrete Fourier Transform of these points is given by:

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

Notice here that we only have as many resultant frequencies as we do data points, so any frequencies present in our data that are higher than the one given by $k = N - 1$ will be undetectable. In our case, we have enough data for the first 64 frequencies, and we don't use the Discrete Fourier Transform but rather the Fast Fourier Transform, which is the same thing, except it's more complicated and is much quicker to compute. It is important to note that in order for Fourier Transforms to work we must have signals that are 2π periodic. Because of this, we need to rescale our domain when plotting in Fourier Space by $\frac{2\pi}{2L}$ where $2L$ is the length of our spatial domain. It is also important to note that for this project we are working with three dimensional data, whereas the theory presented thus far applies specifically to one dimensional data. We will still be using the same concepts that one dimensional Fourier Transforms rely on, but the actual formulas will be

different. Fortunately for us, MATLAB's fftn command computes multidimensional Fourier Transforms for us, so we don't have to implement them on our own.

In this project we will rely on averaging in order to denoise our data. Averaging means that we will take data at each point in time and average all of it together in hopes of random noise cancelling out and leaving us with only the underlying signal. If we only average the data given to us in the time domain, however, we will be unsuccessful because the submarine that is emitting a frequency is moving through space, so the underlying spikes in our data that we are searching for will be in a different place at every time. We can work around this by transforming our data to the frequency domain and averaging it there. When we are in the frequency domain, the underlying spikes in our data will be at the same place every time, since the submarine always emits the same frequency. Averaging will allow us to determine the frequency emitted by the submarine, but it will not tell us where the submarine is at different points in time.

To determine where the submarine is we will use filtering in order to denoise our data. The idea behind filtering is that if we know what frequency our submarine emits, we can transform spatial data from a single point in time into frequency data and then multiply all frequency amplitudes that are not the frequency that we are interested in by a value close to zero and multiply the frequency amplitude that we need by a value close to one. After doing this, we can take an inverse Fourier Transform back into the spatial domain, and since we flattened all useless frequencies, we will now see spikes in our spatial domain at places that are actually emitting the sound that we are looking for, and hence we will be able to track the submarine.

3 Algorithm Implementation and Development

The first algorithm that we had to develop was an algorithm to average the signal in the frequency domain, which required the following steps:

1. Load data and reshape it into a 64x64x64 matrix for all 49 measurements
2. Initialize an empty 64x64x64 matrix that will be used to contain the averaged data
3. Take the 3D Fourier Transform of each data matrix and add it to the matrix containing the averaged data.
4. Take the absolute value of the resultant matrix and divide it by 49 so that it becomes an average of all of the data in the frequency domain.

The implementation of this algorithm is as follows:

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

n = 64; % Fourier modes
samples = 49;

%% averaging signal
ave = zeros(n,n,n);
for i=1:samples
    Un(:,:,:)=reshape(subdata(:,i),n,n,n);
    Unt=fftshift(fftn(Un(:,:,:)));
    ave = ave + Unt;
end
ave = abs(ave)/samples;
```

We can then plot this averaged frequency in order to locate the area around which our center frequency occurs. See figure 1. We can visually inspect the highlighted portion of the plot in order to determine the center in the x, y, and z directions as well as the width in these dimensions. This information leads us to our next algorithm, development of a filter. For this filter, we used a 3D gaussian:

1. define parameters based on plot of averaged signal. We could write a script to generate these parameters, but for the purposes of this project we did not have to.

2. define filter as a 3D gaussian based on the determined parameters

The implementation of this is as follows:

```
x0 = 4.9; %x offset
y0 = -7; %y offset
z0 = 2; %z offset
xs = 0.6; %x std. dev
ys = 0.6; %y std. dev
zs = 0.6; %z std. dev
filter = exp(-((Kx-x0).^2/(2*xs^2)+(Ky-y0).^2/(2*ys^2)+(Kz-z0).^2/(2*zs^2))); %gaussian
```

where Kx, Ky, and Kz have been defined as the X, Y, and Z components of our frequency domain.

We can now apply this filter to each data matrix with the following steps:

1. Take the 3d Fourier Transform of each data matrix

2. Multiply the resultant matrix by our filter to filter out as much noise as possible

3. Take the inverse Fourier Transform of this matrix to return to the time domain

4. Determine the maximum value in this final matrix.

5. Use the indices of that value to select values from our x, y, and z arrays that span our spatial domain. The values extracted from these arrays form the coordinates of points in space where the submarine is located at each point in time.

6. record each of these points in an array. The resulting array contains all points on the submarine's path. See the plotted path in Figure 2. See the x-y coordinates that we should send the subtracking aircraft to in table 1

The implementation of this algorithm is as follows:

```
L = 10;
x2 = linspace(-L,L,n+1); x = x2(1:n); y =x; z = x;
path = [];
for i = 1:samples
    Un(:,:,:)=reshape(subdata(:,i),n,n,n);
    Unt=fftshift(fftn(Un(:,:,:)));
    Untf=Unt.*filter;
    Unf = ifftn(Untf);
    [M] = max(abs(Unf),[],'all');
    [xCoord, yCoord, zCoord] = ind2sub(size(Unf), find(abs(Unf) == M));
    path = [path, [xCoord; yCoord; zCoord]];
end
```

4 Computational Results

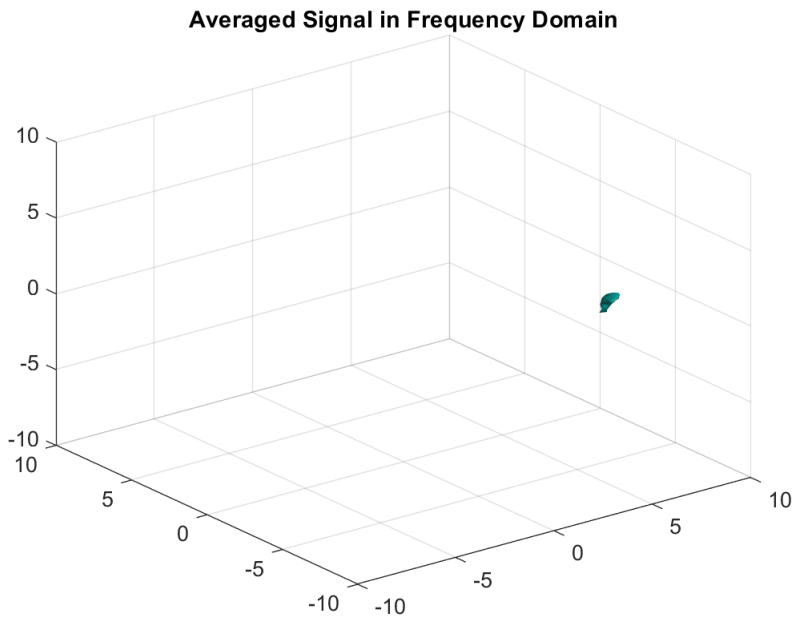

Figure 1: Here we use MATLAB's isosurface command to highlight all of the points in our frequency domain with a value that is at least 70% of the maximum value. Upon inspection of this plot we can determine the spot at which our center frequency occurs and construct a filter accordingly.

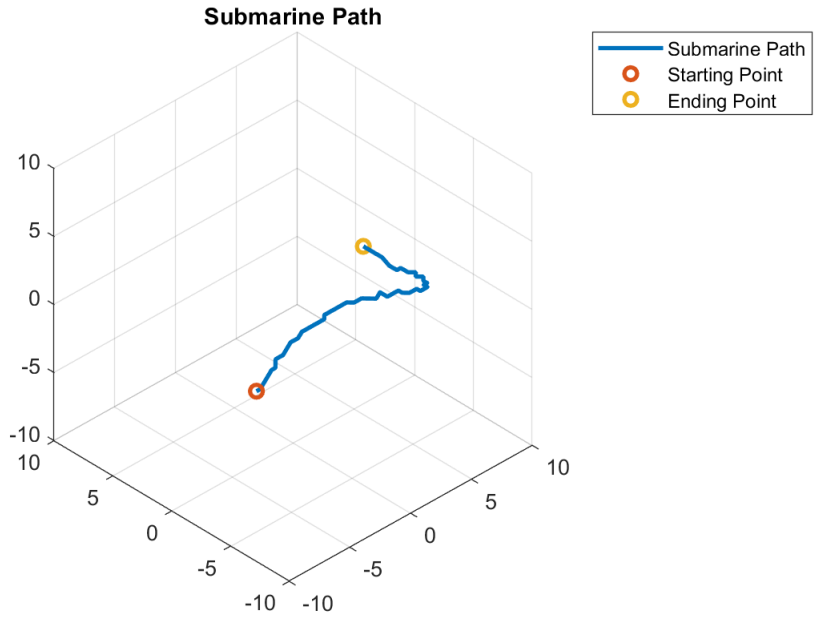

Figure 2: Here we can see the path of the submarine over the two days on which our data was collected.

Time (minutes)	Location
0	(0, 3.125)
30	(0.3125, 3.125)
60	(0.625, 3.125)
90	(1.25, 3.125)
120	(1.5625, 3.125)
150	(1.875, 3.4375)
180	(2.1875, 3.125)
210	(2.5, 3.125)
240	(2.8125, 3.125)
270	(3.125, 2.8125)
300	(3.4375, 2.8125)
330	(3.75, 2.5)
360	(4.0625, 2.1875)
390	(4.375, 1.875)
420	(4.375, 1.875)
450	(4.6875, 1.5625)
480	(5, 1.25)
510	(5.3125, 0.9375)
540	(5.3125, 0.3125)
570	(5.625, 0)
600	(5.625, -0.625)
630	(5.9375, -0.9375)
660	(5.9375, -1.25)
690	(5.9375, -1.875)
720	(6.25, -2.1875)
750	(5.9375, -2.8125)
780	(5.9375, -3.125)
810	(5.9375, -3.75)
840	(5.9375, -4.0625)
870	(5.9375, -4.375)
900	(5.625, -5)
930	(5.625, -5.3125)
960	(5.3125, -5.625)
990	(5.3125, -5.9375)
1020	(5, -5.9375)
1050	(4.6875, -6.25)
1080	(4.6875, -6.5625)
1110	(4.375, -6.875)
1140	(4.0625, -6.875)
1170	(4.0625, -6.875)
1200	(3.4375, -6.875)
1230	(3.4375, -6.875)
1260	(2.8125, -6.875)
1290	(2.5, -6.5625)
1320	(2.1875, -6.5625)
1350	(1.875, -6.25)
1380	(1.5625, -5.9375)
1410	(1.25, -5.625)
1440	(0.9375, -5)

Table 1: X-Y Coordinates of Submarine at all Time Points

5 Summary and Conclusions

For this project we used noisy acoustic data to track a submarine as it moved through the ocean over time. We first denoised the data by taking the 3D Fourier Transform of each data matrix and averaging all of the resultant matrices. We then plotted the averaged signal in the frequency domain in order to determine the center frequency and to develop a filter. Once we developed a working filter, we looped over the noisy data again, took the 3D Fourier Transform of each matrix, multiplied the result by the filter, and then took the inverse 3D Fourier Transform. This gave us data on where in our spatial domain the center frequency was being emitted most strongly at a given point in time. Repeating this process for all data matrices for all points in time allowed us to plot the path of the submarine over time, the plot of which can be seen in figure 2. The path did not look perfectly smooth, which suggests that we were unable to filter out all of the noise, but we filtered out enough that we were able to get a good idea of where the submarine was. If we had had more than 49 data points or if the data was less noisy, then we would have been able to obtain a smoother and more accurate path.
After the path of the submarine had been determined, we were able to determine the x-y coordinates at which a P-8 Poseidon subtracking aircraft should hover over the surface of the ocean in order to always be directly above the submarine. We did this by extracting the x and y components of each point in the submarine path. These coordinates as well as their corresponding times can be seen in table 1.

6 References

Bramburger, Jason. "Computational Methods for Data Analysis." AMATH 482. Lecture, Seattle, University of Washington.

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.
- `ks = fftshift(k)` takes an array as input and returns a new array with the first half and second half swapped. This is needed since when MATLAB performs a Fourier Transform it returns an array with each half of the array on the opposite side that we would expect it on. For multidimensional arrays this command does something similar.
- `Un(:,:,:)=reshape(subdata(:,i),n,n,n);` takes an array as input and returns a new array with the same data but reshaped into the specified dimensions.
- `fftn(Un(:,:,:))` returns an array of the same dimensions as the input with new data representing the multidimensional Fourier Transform of the input array. The output array needs to be passed as input to fftshift in order for us to be able to use it.
- `ind2sub(size(Unf), find(abs(Unf) == M));` returns the indices of the point in the Unf matrix that satisfies `abs(Unf)==M`
- `isosurface(Kx,Ky,Kz,abs(ave)/M,0.7); drawnow` Takes in a 3D domain defined by the first 3 parameters then populates values in this domain with the values specified in the fourth parameter. drawnow then draws this domain and only highlights values that are 0.7 (fifth parameter to isosurface) times as large as the largest value.

Appendix B MATLAB Code

```
clear all; close all; clc;
load subdata.mat % Imports the data as 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);
samples = 49;

%% averaging signal
ave = zeros(n,n,n);
for i=1:samples
    Un(:,:,:)=reshape(subdata(:,i),n,n,n);
    Unt=fftshift(fftn(Un(:,:,:)));
    ave = ave + Unt;
end
ave = abs(ave)/samples;
%% plotting averaged signal in frequency domain
M = max(abs(ave),[],'all');
[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);
figure()
isosurface(Kx,Ky,Kz,abs(ave)/M,0.7)
axis([-10 10 -10 10 -10 10]), grid on,
title("Averaged Signal in Frequency Domain")
drawnow
% x ranges from 4.4 to 5.4, y ranges from -6.5 to -7.5, z ranges from 1.5 to 2.5
%% filtering
x0 = 4.9; %x offset
y0 = -7; %y offset
z0 = 2; %z offset
xs = 0.6; %x std. dev
ys = 0.6; %y std. dev
zs = 0.6; %z std. dev
filter = exp(-((Kx-x0).^2/(2*xs^2)+(Ky-y0).^2/(2*ys^2)+(Kz-z0).^2/(2*zs^2))); %gaussian
%% filtering non-averaged data and inverse transforming back to time domain
path = [];
for i = 1:samples
    Un(:,:,:)=reshape(subdata(:,i),n,n,n);
    Unt=fftshift(fftn(Un(:,:,:)));
    Untf=Unt.*filter;
    Unf = ifftn(Untf);
    [M] = max(abs(Unf),[],'all');
    [xCoord, yCoord, zCoord] = ind2sub(size(Unf), find(abs(Unf) == M));
    xCoord = x(xCoord); yCoord = y(yCoord); zCoord = z(zCoord);
    path = [path, [xCoord; yCoord; zCoord]];
end

%% Plotting submarine path
plot3(path(1,:),path(2,:),path(3,:), "LineWidth", 2);
hold on;
plot3(path(1,1),path(2,1),path(3,1), 'o', 'LineWidth', 2);
```

```
plot3(path(1,end),path(2,end),path(3,end), 'o', 'LineWidth', 2);
axis([-L L -L L -L L])
grid on
legend("Submarine Path", "Starting Point", "Ending Point")
title("Submarine Path")

%% Output subtracking path
for i = 1:samples
    xVal = path(1,i);
    yVal = path(2,i);
    fprintf('%i & (%g, %g)\\\\\n',(i-1)*30,xVal,yVal);
end
```