AMATH 482 Homework 1

Maxime Didascalou

January 27, 2020

Abstract

This project covers the clean up of 3-D grids of acoustic data of the Puget Sound area over a 24 hour period which initially contain a lot of noise. I used MATLAB for Fast Fourier transforms, and filtering and averaging in frequency space in order to determine the path of a submarine located in the grids. After plotting, one can see that it was moving in an upwards spiral.

1 Introduction and Overview

Any sort of physical data collected from the real world will always have some degree of inaccuracy or like in this case, white "noise" that clouds it's useful components. The aim of this project is to find a way to extract this useful information. The end goal is to be able to track or find the path of a moving submarine in the Puget Sound using noisy acoustic data, and then use the results to give the coordinates to a P-8 Poseidon subtracking aircraft so that it can follow the submarine.
The report begins with some theoretical background that is required to understand what the code does, and then explains the algorithms that were used to get the results.

2 Theoretical Background

As we learned from our lectures [1], Fourier introduced the concept of the Fourier transform of a function $f(x)$ to convert it to a function of frequencies:

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

And its inverse takes a function $\hat{f}(k)$ and converts it to a function of space (or time):

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

This should make some sense because of Euler's formula:

$$e^{ix} = \cos x + i \sin x \tag{3}$$

This tells us that the Fourier transform is a function of sins and cosines that represent the different frequencies.
In practice, this is useful for cleaning up a function that has a lot of white noise or unwanted fluctuations because one can convert it to a function of frequencies using the Fourier transform and then apply a filter to it, such as a Gaussian [2]:

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

where τ regulates the width of the filter and k_0 is the central frequency that you want to keep. This essentially focuses on that central frequency and damps everything else. You can then use the inverse Fourier transform on the filtered function to get a cleaner result.
However, in most cases you don't already know what k_0 to use before hand. But through averaging over

several instances of the same repeating noisy function or signal in frequency space, the white noise will cancel out after enough iterations thanks to the law of large numbers, which leaves just the important frequency information [3].
In practice however, we mostly have discrete data points rather than functions to work with. Given a sequence of N equally spaced points $\{x_0, x_1, \ldots, x_{N-1}\}$, we can use the discrete Fourier transform (DFT) [1]:

$$\hat{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi i k n}{N}}. \tag{5}$$

This however is restricted to: $k = -N/2, -N/2+1, \ldots, -1, 0, 1, \ldots, N/2-1$.
The built in MATLAB function `fft` uses the fast Fourier Transform (FFT) which essentially cuts in half the vector of data points which gives 2 smaller DFTs, and then the process is repeated untill the vectors can't be cut in half anymore. This reduces the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

3 Algorithm Implementation and Development

The data is over 49 iterations (49 different times) which is why the for-loops go from 1 to 49.

This project had three main computational parts:

1. Setting up and visualizing the problem.
2. Finding the central frequency of the submarine through averaging of the spectrum.
3. Using the central frequency to filter out the white noise and figure out the path of the submarine.

Algorithm 1: Import and set up variables

Import data from subdata.mat
Set up spacial coordinates
Set up frequencies according to the requirements of the FFT
Scale them by $\frac{2\pi}{L}$ since the FFT assumes 2π periodic signals, where L is the spacial domain
for $j = 1 : 49$ **do**
  Assign the jth column of subdata to be a 64 by 64 by 64 3D Matrix
  Plot the matrix to visualize the problem
end for

Algorithm 2: Finding frequency signature

Set frequency signature to be a 64 by 64 by 64 matrix of zeroes
for $j = 1 : 49$ **do**
  Take 3D Fourrier transform of jth matrix from Algorithm 1
  Shift the returned values to the conventional format (the built in MATLAB Fourier transform returns the values for the negative frequencies last)
  Add the shifted matrix to frequency signature
end for
Divide frequency signature by 49
Find central frequency by finding the frequency coordinates of the max of frequency signature

Algorithm 3: Finding coordinates

Construct a 3D filter around central frequency found in Algorithm 2
Set coordinates to be a 49 by 3 matrix of zeroes
for $j = 1 : 49$ **do**
  Multiply jth shifted matrix from Algorithm 2 by filter, and shift it back to the MATLAB conventions
  Take the inverse Fourier transform of the result
  Find the coordinates of it's max (coordinates of sub at time j)
end for
Plot path of submarine using coordinates

4 Computational Results

Figure 1 shows the path of the submarine, and Table 1 shows the coordinates of the submarine at every moment in time.
The frequency signature found in Algorithm 2 is: $(k_x, k_y, k_z) = (5.3407, -6.9115, 2.1991)$.

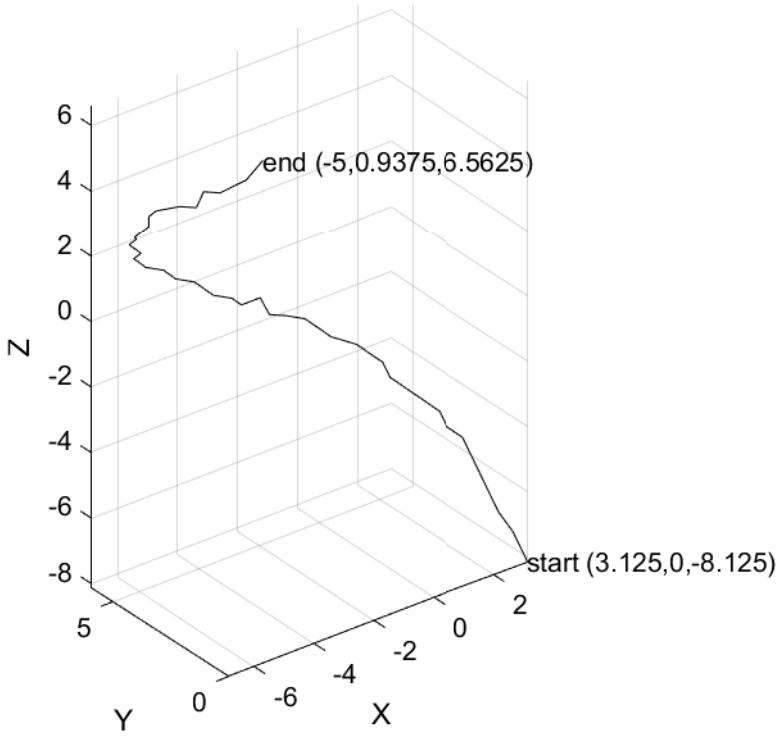

Figure 1: Path of submarine

time	x	y
1	3.1250	0
2	3.1250	0.3125
3	3.1250	0.6250
4	3.1250	1.2500
5	3.1250	1.5625
6	3.1250	1.8750
7	3.1250	2.1875
8	3.1250	2.5000
9	3.1250	2.8125
10	2.8125	3.1250
11	2.8125	3.4375
12	2.5000	3.7500
13	2.1875	4.0625
14	1.8750	4.3750
15	1.8750	4.6875
16	1.5625	5.0000
17	1.2500	5.0000
18	0.6250	5.3125
19	0.3125	5.3125
20	0	5.6250
21	-0.6250	5.6250
22	-0.9375	5.9375
23	-1.2500	5.9375
24	-1.8750	5.9375
25	-2.1875	5.9375
26	-2.8125	5.9375
27	-3.1250	5.9375
28	-3.4375	5.9375
29	-4.0625	5.9375
30	-4.3750	5.9375
31	-4.6875	5.6250
32	-5.3125	5.6250
33	-5.6250	5.3125
34	-5.9375	5.3125
35	-5.9375	5.0000
36	-6.2500	5.0000
37	-6.5625	4.6875
38	-6.5625	4.3750
39	-6.8750	4.0625
40	-6.8750	3.7500
41	-6.8750	3.4375
42	-6.8750	3.4375
43	-6.8750	3.1250
44	-6.5625	2.5000
45	-6.2500	2.1875
46	-6.2500	1.8750
47	-5.9375	1.5625
48	-5.3125	1.2500
49	-5.0000	0.9375

Table 1: Coordinates for the aircraft to follow

5 Summary and Conclusions

After summing over all the iterations of the data in frequency space (Fourier transform), I was able to identify a central frequency that was common in all the iterations, which made it possible for me to go back and filter each iteration in frequency space around that frequency (in order to get rid of the unwanted noise), and then use the inverse Fourier transform to get a cleaned up version of the original data. This new version made it possible to identify the positioning of the submarine at each iteration, and thus plot its path. This allowed me to see that the submarine was moving in an upwards spiral.

Also, since we now know the central frequency, we don't have to go through the process of averaging anymore. And if more data keeps coming in, all we would need to do it take the Fourier transform of that data, filter it around the central frequency, and then take the inverse Fourier transform to find the submarines location and have a P-8 poseidon subtracking aircraft follow it in real time.

References

[1] Jason J. Bramburger *Lecture 1: Basics of Fourier Series and the Fourier Transform.* University of Washington, Lecture Notes, 2021.

[2] Jason J. Bramburger *Lecture 2: Radar Detection and Filtering.* University of Washington, Lecture Notes, 2021.

[3] Jason J. Bramburger *Lecture 3: Radar Detection and Averaging.* University of Washington, Lecture Notes, 2021.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y,Z] = meshgrid(x,y,z)` returns 3-D grid coordinates defined by the vectors `x`, `y`, and `z`. The grid represented by `X`, `Y`, and `Z` has size `length(y)`-by-`length(x)`-by-`length(z)`.
- `Y = fftn(X)` returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D transform is equivalent to computing the 1-D transform along each dimension of `X`. The output `Y` is the same size as `X`.
- `X = ifftn(Y)` returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D inverse transform is equivalent to computing the 1-D inverse transform along each dimension of `Y`. The output `X` is the same size as `Y`.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `X = ifftshift` undoes the result of `fftshift`.
- `M = max(A,[],'all')` finds the maximum over all elements of `A`.
- `[M,I] = max(A,[],_,'linear')` returns the linear index into `A` that corresponds to the maximum value in `A`.
- `[I1,I2,...,In] = ind2sub(sz,ind)` returns `n` arrays `I1,I2,...,In` containing the equivalent multidimensional subscripts corresponding to the linear indices `ind` for a multidimensional array of size `sz`. Here `sz` is a vector with `n` elements that specifies the size of each array dimension.
- `plot3(X,Y,Z)` plots coordinates in 3-D space.

Appendix B MATLAB Code

```
load subdata.mat

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);

for j=1:49
    U_noisy(:,:,:,j)=reshape(subdata(:,j),n,n,n);
    M = max(abs(U_noisy),[],'all');
%     close all, isosurface(X,Y,Z,abs(U_noisy)/M,0.5)
%     axis([-20 20 -20 20 -20 20]), grid on, drawnow
%     pause(0.1)
end
```

Listing 1: Algorithm 1

```
%averaging in frequency space
U_average_freq = zeros(n,n,n);
for j=1:49
    U_freq_noisy(:,:,:,j) = fftshift(fftn(U_noisy(:,:,:,j)));
    U_average_freq = U_average_freq + U_freq_noisy(:,:,:,j);
end
U_average_freq = U_average_freq/49;


% finding the highest frequency
[m,ind] = max(U_average_freq(:));
[ind_kx,ind_ky,ind_kz] = ind2sub([n,n,n],ind);
center_Kx = Kx(ind_kx,ind_ky,ind_kz);
center_Ky = Ky(ind_kx,ind_ky,ind_kz);
center_Kz = Kz(ind_kx,ind_ky,ind_kz);
```

Listing 2: Algorithm 2

```
%setting up the filter
tau = 0.3;
filter = exp(-tau.*((Kx - center_Kx).^2+(Ky - center_Ky).^2 + (Kz - center_Kz).^2));

coordinates = zeros([49 3]);

for j=1:49
   % filtering the signal
    U_freq_filtered = ifftshift(U_freq_noisy(:,:,:,j).*filter);
    U_filtered = ifftn(U_freq_filtered);

    % finding the x and y coordinates for the plane to follow
    [m,ind] = max(U_filtered(:));
    [ind_x,ind_y,ind_z] = ind2sub([n,n,n],ind);
    coordinates(j,1) = X(ind_x,ind_y,ind_z);
    coordinates(j,2) = Y(ind_x,ind_y,ind_z);
    coordinates(j,3) = Z(ind_x,ind_y,ind_z);

    % plotting
%     M = max(abs(U_filtered),[],'all');
%     close all, isosurface(X,Y,Z,abs(U_filtered)/M,0.5)
%     axis([-20 20 -20 20 -20 20]), grid on, drawnow
%     pause(1)
end
plot3(coordinates(:,1), coordinates(:,2),coordinates(:,3)) %plotting path
axis equal
xlabel('X')
ylabel('Y')
zlabel('Z')
stringstart = ['start (',num2str(coordinates(1,1)),',',
               num2str(coordinates(1,2)),',',num2str(coordinates(1,3)),')']
stringend = ['end (',num2str(coordinates(end,1)),',',
             num2str(coordinates(end,2)),',',num2str(coordinates(end,3)),')']
text(coordinates(1,1), coordinates(1,2),coordinates(1,3),stringstart)
text(coordinates(end,1), coordinates(end,2),coordinates(end,3),stringend)
grid on
```

Listing 3: Algorithm 3