AMATH 482
Computational Methods for Data Analysis

Zach Shaw

Winter 2019

Homework 1

Due: January 25, 2019.

January 24, 2019

0.1 Abstract

This document details how I used Fourier Transforms and Matlab to average noisy ultrasound data taken at 20 points in time. Using 3D filtering, I was able to clean up the data, identify the trajectory of the marble, plot the trajectory, and identify where I want to focus acoustic waves to dissolve the marble safely. The methods described can be extended to contain more than just 20 points of data. After completing the project, I overcame my challenge of conceptualizing the 3D aspect of this problem.

1 Introduction and Overview

As dogs, especially puppies, are want to do, my dog has swallowed a marble. I have received 20 samples of ultrasound data for my dog's intestines. I have to identify the frequency of the marble, filter the ultrasound data around that frequency, and then identify the location to focus an intense acoustic wave to dissolve the marble.

In order to identify the frequency of the marble, I need to Fast Fourier Transform the data, average the transformed data over all 20 samples, calculate the max and index of the shifted average, and get the wave numbers, represented by k_x, k_y, k_z. With the location of the marble's frequency, I will next construct a 3D Gaussian filter to de-noise the data. Once the data is filtered in Fourier space, I need to return it to the time domain, reshape it back to $20x64^3$, calculate the maximum, and break it down into the x, y, z components of the path. Then the final location is just the last x, y, z component.

2 Theoretical Background

At the heart of this problem, it is a 3D signal processing problem with a frequency moving in time and space. In order to solve this problem, we will need to make extensive use of the Fast Fourier Transform (FFT). The FFT is a function that efficiently, ($O(n\log(n))$), computes the Fourier Transform of a variable. A consequence of this algorithm is that our discretization has to have 2^n points. The Fourier Transform is defined as:

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

There is also an inverse Fourier Transform (ifft) defined as:

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

The difficulty with the bounds is the first thing we must overcome. FFT approximates the bounds, instead of $(-\infty, \infty)$, FFT uses $(-\pi, \pi)$. This works because the FFT expands a function in terms of sine and cosine, and those functions are periodic on the domain $(-\pi, \pi)$. We then need to re-scale our wave numbers, so that it they apply to a general domain $(-L, L)$. To do this, use line 15 from the Appendix B. The other thing of note arises from the periodicity of FFT. The point at the beginning of the domain is going to be equal to the point at the end of our domain. To overcome this, I use $n+1$ points when using the linspace function. Then define my domain as the linspace(1:n). This gets all of the unique points within the domain. The final thing to note is that the FFT algorithm shifts the output. It outputs the results on the domain $(\pi, -\pi)$.

The reason the first step is to average the data over the 20 samples is because that is a common filtering technique. The random noise tends towards 0 when you average, only leaving the signal. Now that we understand the caveats of the functions we are going to use and why we are doing this, lets build the algorithm.

3 Algorithm Implementation and Development

3.1 Step 1: Average the Data

The first step is to average our noisy data over the 20 samples to calculate the maximum frequency. If you look at figure 1, it is very difficult to determine if there is any underlying frequency. To start, create an $nxnxn$ matrix to store the average, then create a loop from 1 to the number of samples, grab each individual sample of data, and add it together in the newly created average matrix. Once the loop is complete, divide by the number of samples find the maximum and the index of the maximum. After obtaining the index, use the ind2sub function to get the wave numbers k_x, k_y, k_z that represent the location of the central frequency. Note: these are the locations of the linear indices, we still need to collect the actual wave numbers from the ks vector.

3.2 Step 2: Filter the Data

Now that we have the location of our central frequency, we can filter our data. First, lets construct a 3D Gaussian filter. A 3D Gaussian looks like:

$$e^{-(x^2+y^2+z^2)} \tag{3}$$

But we only know our central frequency in Fourier space, and we are not centered at the origin, so our Gaussian filter is going to be of the form:

$$e^{-\tau((KX-ks(ky))^2+(KY-ks(kx))^2+(KZ-ks(kz))^2)} \tag{4}$$

Where τ is the bandwidth of the filter, KX, KY, KZ represent the grip of k values for the corresponding dimension, and $ks(k)$ is the location of the central frequency in the corresponding dimension. Note: my KX and KY got flipped, so I swapped the kx and ky. I chose my bandwidth to be $\tau = .2$. Remember that this filter is shifted, shift it again so that way it is compatible with the FFT of the noisy data. Now, construct the loop detailed by lines 53-65. This takes a $64x64x64$ grid from each time sample, fftn shifts the sample, filters it, and then puts it back in the time domain. Then, isosurface attempts to detect a path contained within the data. The blue spheres in figure 2 show the position of the marble as time progresses. And finally, the loop calculates the maximum value and index of the filtered data. The algorithm then takes that index in the X, Y, Z grid and stores it in the location matrix to plot the trajectory of the marble.

3.3 Step 3: Identify Position at $t = 20$

Now that we have constructed the location variable, simply run line 74 to get the ordered triplet. This gives the location where we should focus an intense acoustic signal to dissolve the marble to save the dog.

4 Computational Results

The data given was complete noise, there was no way to discern if there was any relevant information contained within the data. The filtering algorithm I implemented was able to filter the data, and extract the path of the marble within the dog. The unfiltered data is shown in figure 1, and the trajectory of the marble is shown in figure 2. A simple call of

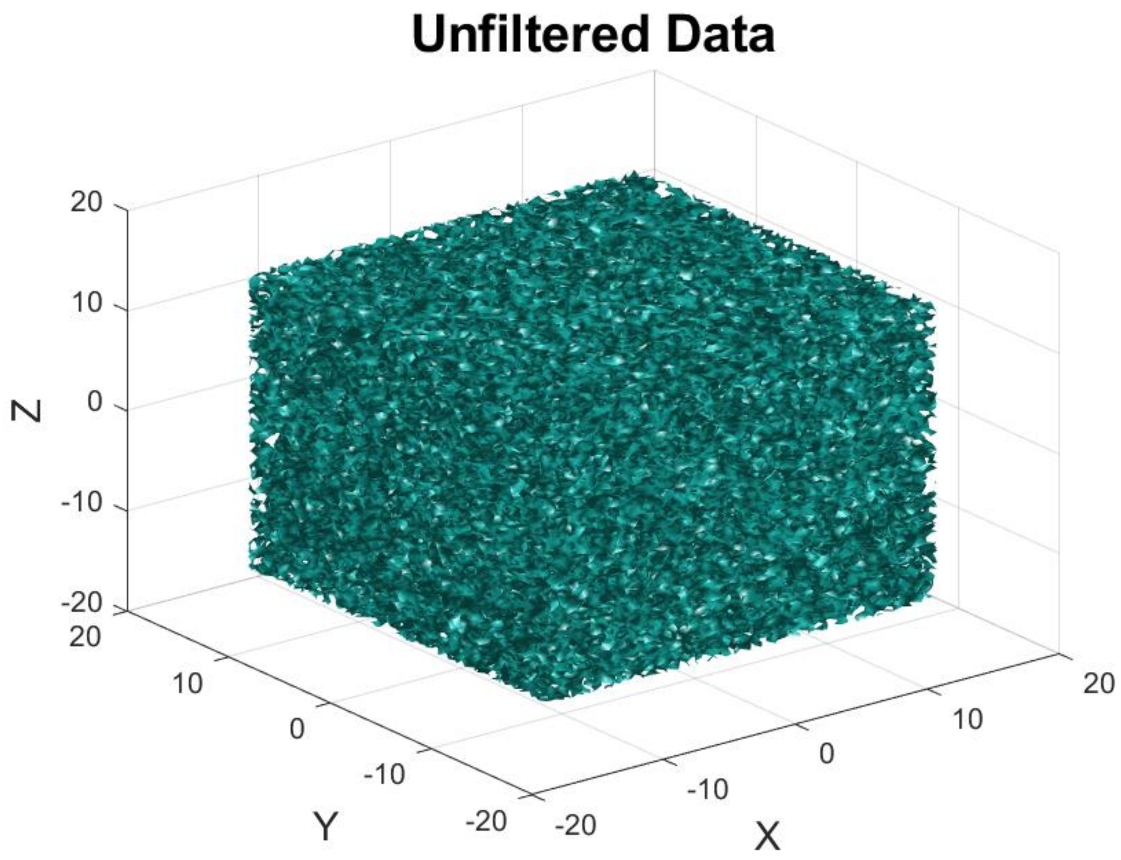

Figure 1: The data before filtering for the signal

my location variable at $t = 20$, this is like $\vec{r}(t = 20) < x(20), y(20), z(20) >$, shows that an intense acoustic wave should be focused at the location: $\boxed{(-5.625, 4.21875, -6.09375)}$.

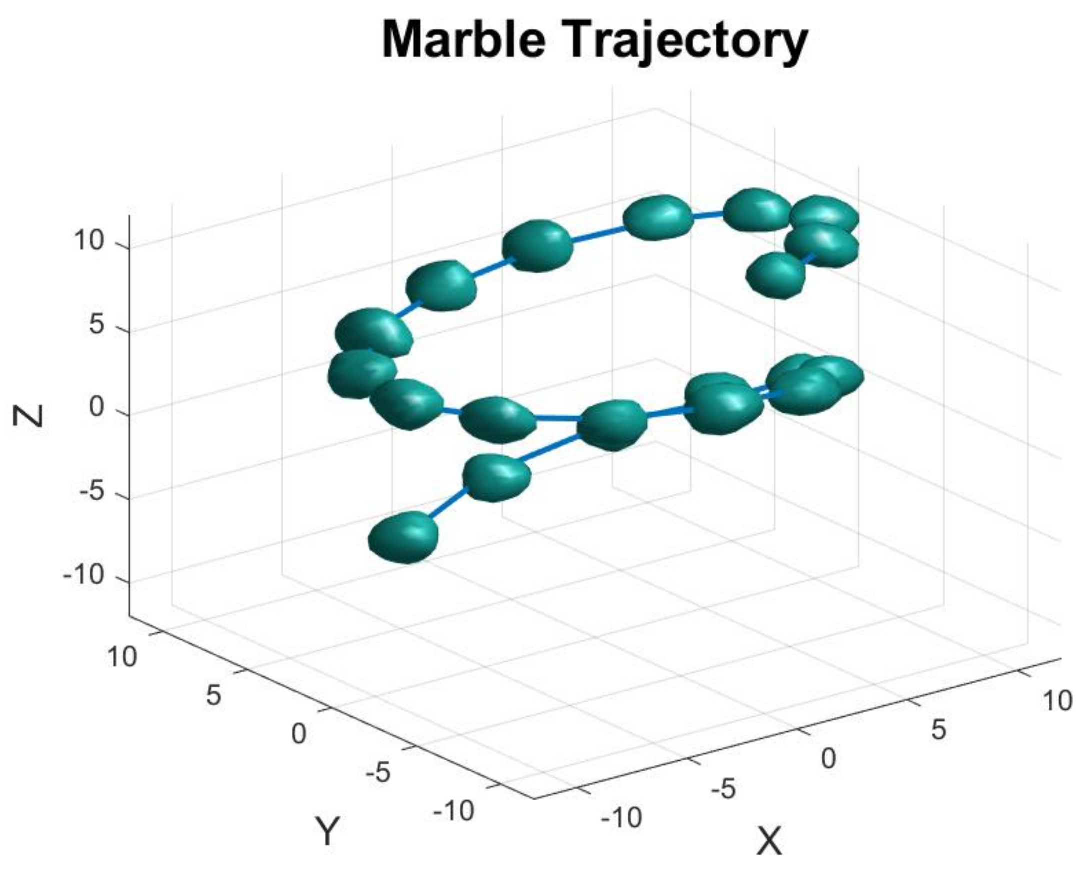

Figure 2: The result of filtering, produced the trajectory of the marble and isosurface shows the path over time, starting from the top of the spiral, and moving downwards.

5 Summary and Conclusions

In retrospect, this was an interesting assignment. My biggest issue was comprehending how to get the $x, y, \text{and} z$ information out of the filtered data. Lines 57 through 60 were used to solve for the $x, y,$ and, z components of the solution. One of the cool things about this project was transforming the 3D data into the Fourier domain because even though the marble was moving, we were able to detect it easily in the Fourier domain.

The methods used to solve this problem can be extended to include more than just 20 samples of data. The important thing is that the samples must contain 2^n data points to maximize the efficiency of FFT.

A MATLAB Functions and Implementation

Functions I used to implement the algorithm and why:

linspace(-L,L,n+1): This function was used to generate a sequence of points from -L to L with n+1 points. We use n+1 points because FFT assumes periodic boundary conditions. So if n was used in the linspace command, then the first point would be the same as the last point, and we would miss a point in the discretization.

fftshift(ks): When generating the fourier nodes, they are generated from positive to 0 and then 0 to negative. fftshift makes it go from negative to 0, and then 0 to negative.

]X Y Z]=meshgrid(x,y,z): The meshgrid function is used to generate a discretization grid.

reshape(Undata(1,:),[n n n]): Reshape takes a long list of data, and reshapes it into the specified size. I used this because the x, y, and z components of the data were stuck together in the measurements spread across 20 samples.

abs(Un): This takes the absolute value of Un. If Un is a complex valued number or matrix, it multiplies by the complex conjugate to calculate the absolute value.
isosurface(X,Y,Z,abs(Un),.4): Isosurface is a function that goes through volumetric data provided by the 4th input variable, plots it against the 3 matrices in the first 3 inputs, and tries to identify isosurface data at the isofurface value described by the 5th input.

axis([-20 20 -20 20 -20 20]): The axis command fixes the minimum and maximum x, y, and z values for the active figure.

x,y,zlabel(...): The x,y, or zlabel function sets the label of the corresponding axis to the specified label.

title(...): The title command sets the title of the active figure.

print(...): This print function saves the active figure to the name specified by the 1st input. The file type is determined by the 2nd input.

zeros(n,n,n): This function is used for creating a matrix of all 0. The dimension of the matrix created is given by the 3 input arguments.

size(Undata,1): This grabs the number of rows in the data. I used this to get each of the 20 samples of data.

[Max,Ind]=max(abs(ave(:))): This finds the value of the center frequency, and the index it occurs.

[kx,ky,kz]=ind2sub([64,64,64],Ind): This ind2sub function takes the index of the max frequency and finds the kx, ky, and kz where the max frequency occurs in 3D.

exp(): creates an exponential function with the provided argument. This is used to filter the data, so we need to have 3 variables centered at the central frequency. These three variables are provided by $(Kx-ks(ky))^2+(Ky-ks(kx))^2+(Kz-ks(kz))^2$. There is also the bandwidth term τ multiplying all 3 variables.

plot3(): plot3 takes parametric data for a line in 3D and plots the data.

B MATLAB Code

```
%Zach Shaw
%1/14/18
%AMATH 482
%Professor Kutz
%Homework 1

clear all; close all; clc;
load Testdata
L=15; % spatial domain
n=64; % Fourier modes
%Define the Coordinate Domain
x2=linspace(-L,L,n+1);
x=x2(1:n); y=x; z=x;
%Define Fourier Nodes
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1];
ks=fftshift(k);
%Create grids in X, Y, and Z
[X,Y,Z]=meshgrid(x,y,z);
%Create Fourier Grids
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);
%Plot the noisy data
Un(:,:,:)=reshape(Undata(1,:),n,n,n);
isosurface(X,Y,Z,abs(Un),.4)
axis([-20 20 -20 20 -20 20])
grid on
xlabel('X','Fontsize',14)
ylabel('Y','Fontsize',14)
zlabel('Z','Fontsize',14)
title('Unfiltered Data', 'Fontsize',18)
print('Unfilt','-djpeg') %Print to file for LaTeX use
%% Part 1
%Average the specturm and determine the frequency signature (
   center
%frequency generated by the marble.

%Average the data
ave=zeros(n,n,n);
%Average through the given time stamps
for jj=1:size(Undata,1)
```

```
    ut(:,:,:)=fftshift(fftn(reshape(Undata(jj,:),n,n,n)));
    ave=ave+ut(:,:,:);
end
fftshift(ave);
ave=ave./size(Undata,1);
[Max, Ind] = max(abs(ave(:))); %Find the Index of the Central
    Frequency
[kx,ky,kz]=ind2sub([64,64,64],Ind); %Find the Fourier Nodes
   that correspond to the central frequency


%Filter the data
tau=.2; %define the bandwidth of the filter
filterF=exp(-tau*((Kx-ks(ky)).^2+(Ky-ks(kx)).^2+(Kz-ks(kz))
   .^2)); % establish filter
filter=fftshift(filterF); %shift the filter so that it is
   compatible with fft of Undata
%Filter all 20 time stamps
for z=1:size(Undata,1)
    U=reshape(Undata(z,:),[n n n]);
    UfF=fftn(U).*filter;%Filter each time stamp
    Uf=ifftn(UfF);%Move the data back to the time domain
    figure(3)
    isosurface(X,Y,Z,abs(Uf),0.4)
    hold on
    axis([-12 12 -12 12 -12 12]), grid on
    [Max,I]=max(Uf(:));%Calculate the maximum value of the
       filtered data
    %The maximum has 3 components (x,y,z), this puts it as an
        ordered
    %triplet
    location(z,:)=[X(I) Y(I) Z(I)];
end
%Plot the set of 20 (x,y,z)'s
plot3(location(:,1),location(:,2),location(:,3),'LineWidth'
   ,2)
xlabel('X','Fontsize',14)
ylabel('Y','Fontsize',14)
zlabel('Z','Fontsize',14)
title('Marble Trajectory','Fontsize',18)
print('Trajec','-djpeg')
%Answer to part 3
Part_3_ans=[location(20,1), location(20,2), location(20,3)]
hold off
```