AMATH 482 HW 1

Zoheb Siddiqui

January 24, 2019

Abstract

In this paper we attempt to de-noise and track a signal in time using Fourier transform. We first average the signal in the Fourier domain to eliminate noise. Then we filter the signal and bring it back in the time domain to track the movement at different instances of time.

Introduction

This is the first homework assignment for AMATH 482. We have a dog who has possibly swallowed a marble and we are trying to determine two things.

- Is there a marble in the dog's stomach
- If so then we must track its position

We have been given a data-set that contains frequency signals reflected back from the intestine of a dog. The data-set contains 20 frequency measurements from the intestine at 20 different times. We need to track the position of the marble as it moves through the intestine if it exists.

Theoretical Background

The signals obtained are extremely noisy and we assume that the noise is distributed normally so averaging the signals would remove the noise.

Since the marble moves around thus, averaging the signal in the time domain wouldn't be helpful as it's position keeps changing.

Thus, we must transform the signal to the Fourier domain and then take the average. Since we are using a Fourier transform, we shall have to scale the wavenumbers by $2 * \pi/L$ as the Fourier domain ranges from $[-\pi\ \pi]$. We shall normalize this average and then if there is a peak we know that there is indeed a marble in the intestine.

If there is a marble then we need to find it's frequency in the Fourier domain and use it to create a filter. We then apply this filter to each frequency measurement in the Fourier domain and then convert it back to the time domain.

For each filtered measurement in the time domain we need to find the coordinates with the maximum frequency. This will be the position of the marble at that instant.

Algorithm Implementation

We implement the solution in 2 parts.

Part 1: First we need to determine if the marble exists. If it does then we need to identify it's frequency and formulate a Gaussian filter using it.

```
Declare a empty 3D matrix to store the average.
For each iteration (ranging from 1 to 20):
    Reshape a row in the data-set to a 64x64x64 3D matrix.
    Find the Fourier transform of this matrix and add it to the variable that stores the
    average.
Normalize the Fourier shift of the average and then plot all the points above a magnitude of
0.6.
```

The plot should display the existence of a marble or the lack there of.

If the marble exists then we extract the frequency emitted by it and use it to create a Gaussian filter. The Gaussian filter is created as $\exp^{-\tau((Kx-\bar{x})^2+(Ky-\bar{y})^2+(Kz-\bar{z})^2)}$. τ represents the width of the filter which can be altered to make the filter narrower or wider. $\bar{x}$, $\bar{y}$, $\bar{z}$ are the frequency signature in the X axis, Y axis and Z axis respectively.

Part 2: Now that we have the Gaussian filter we must track the movement of the marble.

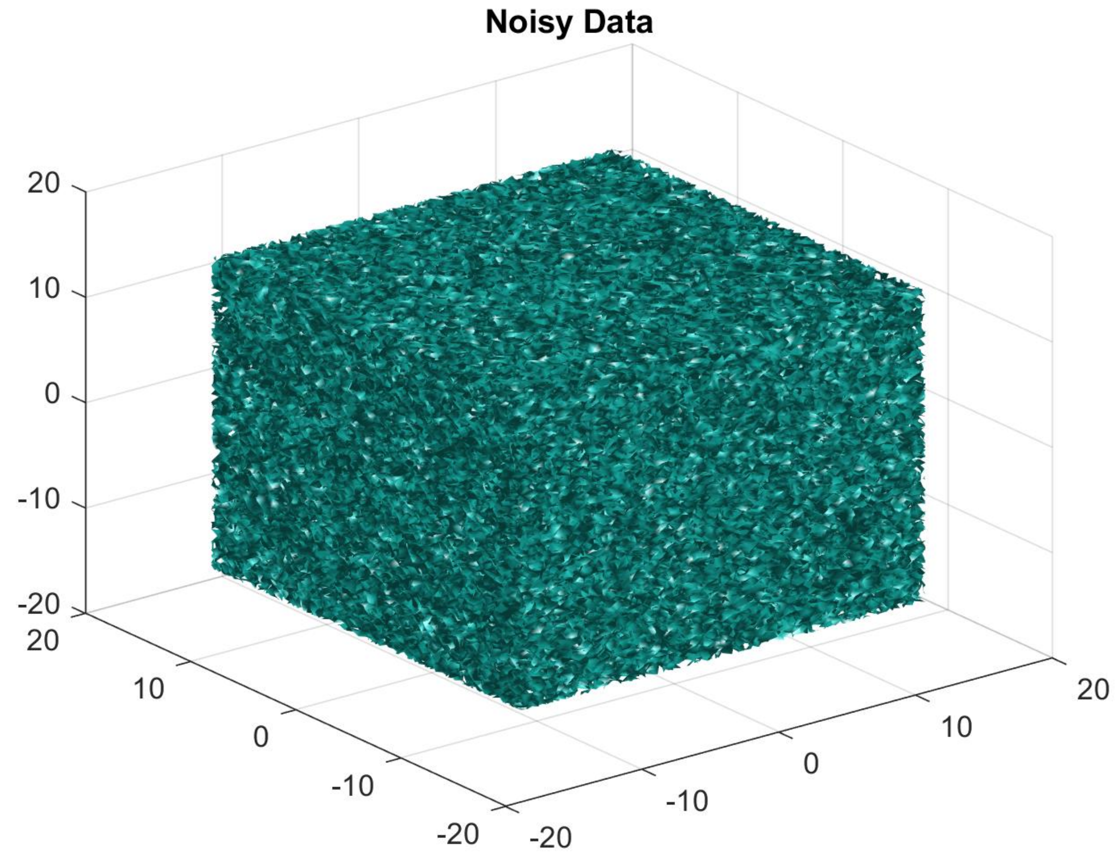
```
For each iteration (ranging from 1 to 20):
    Reshape a row in the data-set to a 64x64x64 3D matrix.
    Fourier transform the reshaped noisy data and apply the Fourier shift function to it.
    Multiply each element in the transform with the filter.
    Compute the inverse Fourier shift of the filtered data and then the inverse Fourier
    transform.
    We now store the maximum value of the filtered data in the time domain to eliminate
    the complex parts.
    The X,Y,Z coordinates corresponding to the maximum of the filtered data represent the
    position of the marble at that instant in time.
```

We plot the coordinates for each time slice. This gives the trajectory of the marble.

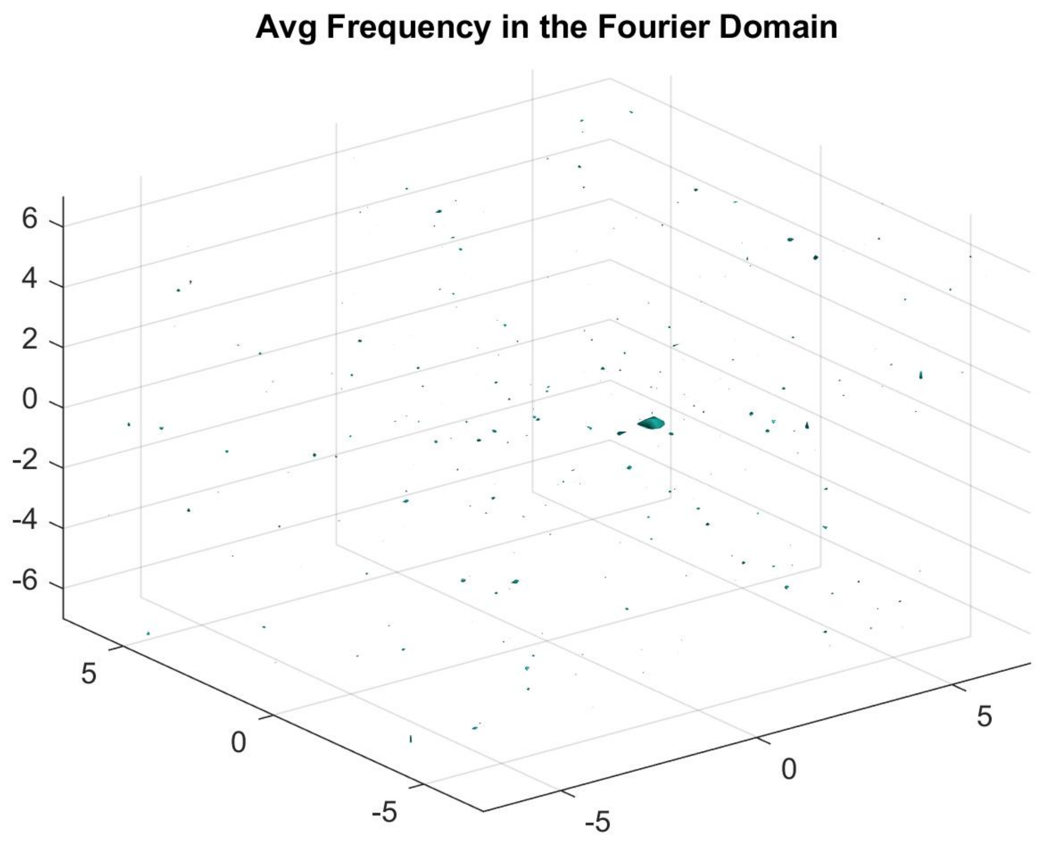
Computational Results

Since the problem was phased in 3 parts we shall present our computational results in 3 parts as well.

The plot of the noisy data in the Fourier domain looks like:

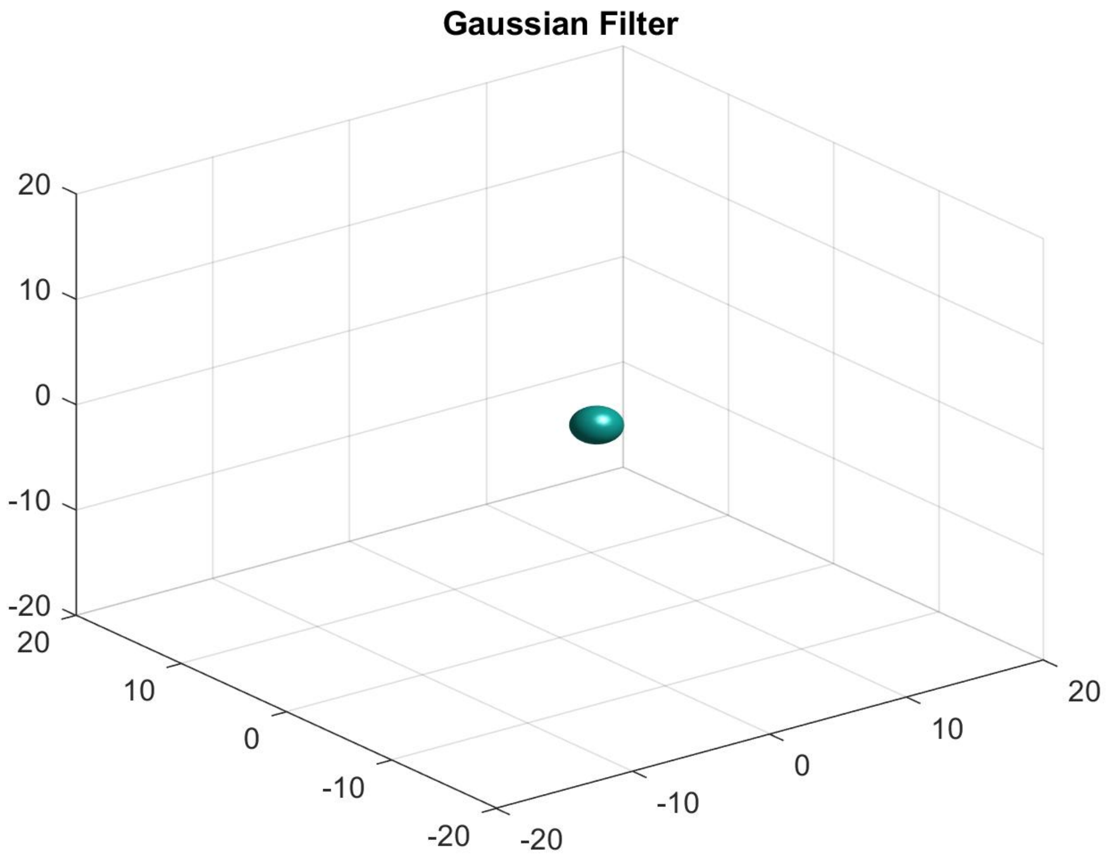


Part 1: Using the algorithm in part one we plot the normalized average of the transformed data. We plot all points above a magnitude of 0.6 and get the following plot.

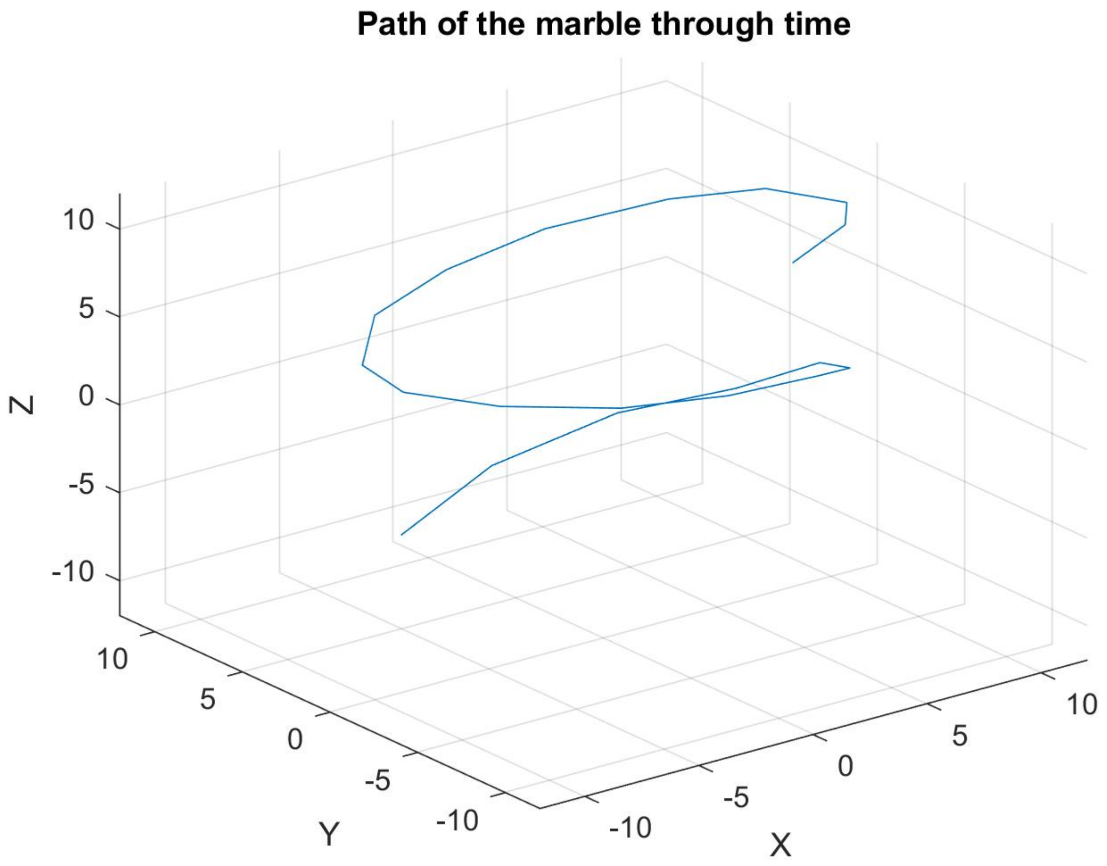


This confirms the existence of the marble. We now find the frequency signature generated by the marble which is [1.8850, -1.0472, 0].

Part 2: Now we compute the Gaussian filter from the frequency signature we extracted in part 1.
In the Fourier domain, the filter looks like this:



Now we apply the filter to the noisy data in the Fourier domain and transform it back to the time domain to extract the path of the marble.
The path of the marble is:



Part 3: Each point in the path represents its position at that instant in time. We need to find the position in the 20th instant which is just the 20th set of coordinates.
The position of the marble at the 20th instance is (-5.6250, 4.2188, -6.0938). This is where the intense acoustic wave needs to be focused to break the marble.

Summary

The problem given to us is that a dog has swallowed a marble and we have frequency measurements from the intestine at 20 instances in time. We need to figure out the location of the marble at the 20th instant.

The solution to the problem consists of 3 parts. First we transform the data to the Fourier domain and average it out to eliminate noise. We check if this averaged transform has a peak denoting the existence of the marble.

We extract the frequency signature of the marble and use it to create a Gaussian filter.

The Gaussian filter is applied to each transformed data-set and the result is transformed back in the time domain. The coordinates of the highest point indicate the position of the marble at that instant in time.

Appendices

MATLAB functions used

fftn() : Takes the Fourier transform of a n dimensional signal in the time domain and converts it to the frequency domain.
fftshift() : Rearranges the Fourier transform passed to it so that the zero-frequency component is in the center.
ifftshift() : Reverses the effect of fftshift()
ifftn() : Inverses the Fourier transform in n dimensions to get back the original signal in the time domain.
reshape() : Reshapes a data structure to different dimensions.
isosurface() : Generates 3D figures from volume data by connecting points of similar contours together.
plot3() : Plots data in 3D.
ind2sub() : Converts indices in a 1D array to an n dimentional array.

Code

```
clear all;close all; clc;

load Testdata

L=15; % spatial domain
n=64; % Fourier modes
t2=linspace(-L,L,n+1);
t=t2(1:n);

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);

%%
%1)
Utn_avg = zeros(1,n);
for j=1:20
```

```
    Un(:,:,:)=reshape(Undata(j,:),n,n,n);
    Utn(:,:,:) = fftn(Un(:,:,:));

    Utn_avg = Utn_avg + Utn(:,:,:);
end

figure(1)
close all, isosurface(X,Y,Z,abs(Un),0.4)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
title('Noisy Data')

Utn_avg_p=fftshift(Utn_avg)/20;
to_plot = abs(Utn_avg_p)/max(max(max(abs(Utn_avg_p))));
figure(2)
isosurface(Kx,Ky,Kz,to_plot,0.6)
axis([-7 7 -7 7 -7 7]), grid on, drawnow
title('Avg Frequency in the Fourier Domain')

[mxv,idx] = max(to_plot(:));
[r,c,p] = ind2sub(size(to_plot),idx); %3d indices
Frequency_signal = [Kx(r,c,p), Ky(r,c,p), Kz(r,c,p)];
disp(Frequency_signal);

%%
%2)
filter = exp(-0.2*( (Kx - Kx(r,c,p)).^2 + ...
(Ky - Ky(r,c,p)).^2+ (Kz - Kz(r,c,p)).^2));

figure(3)
isosurface(Kx,Ky,Kz,abs(filter),0.6)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
title('Gaussian Filter')

coords = zeros(3,20);
for j=1:20
    Un(:,:,:)=reshape(Undata(j,:),n,n,n);
    Utn(:,:,:) = fftn(Un(:,:,:));
    Utn(:,:,:) = fftshift(Utn(:,:,:));
    Utfn(:,:,:) = filter.*Utn(:,:,:);
    ufn(:,:,:) = ifftshift(Utfn(:,:,:));
    ufn(:,:,:) = ifftn(ufn(:,:,:));
    abs_ufn = abs(ufn);
    [mxv,idx] = max(abs_ufn(:));
    [r,c,p] = ind2sub(size(abs_ufn),idx);
    coords(:,j) = [X(r,c,p),Y(r,c,p),Z(r,c,p)];
```

```
end
figure(4)
plot3(coords(1,:),coords(2,:), coords(3,:));
axis([-12 12 -12 12 -12 12]), grid on, drawnow
title('Path of the marble through time')
xlabel('X')
ylabel('Y')
zlabel('Z')
%%
%3)
disp(coords(:,20));
```