An Ultrasound Problem

By: Priyanshi Rastogi

1. Abstract:

In this report we are trying to locate and find the trajectory of an object using Fourier Transform and filtering. We are able to find useful information by reducing noise (by filtering) and using the average of the given signals. We use Fourier Transform to transform our given time signals into frequency signals and then take the average of that and de-noise it by applying a filter (Gin). We are then able to track an object's location and path, by taking the inverse Fourier Transform of the filtered average frequency signal, in time (Gin).

2. Introduction/Overview:

Fluffy, our precious dog, has swallowed a marble. Fluffy moves a lot and thus the measurements of his ultrasound have a lot of noise in them. We want to be able to find out where the marble is located to break the marble and save our dog Fluffy.

We are given 20 different measurements each with a random frequency at points. In order to reduce that, we average together all of the frequency signals (transform the time signals given to us) so that the random frequencies go away (are less focused on) and we are left with the more important data, the frequencies created by the marble. We can further clear up the data by applying a gaussian filter centered near the center frequency to get the true signal we want. Once we have this, we apply an inverse transform to convert the cleaned-up frequency signal back into a time signal and therefore are able to see the path of the marble and where it's located at each measurement.

3. Theoretical Background:

We are given 20 different measurements of the time signal through Fluffy's ultrasound. Taking the **F*ourier Transform*** of this signal gives us a frequency signal that can further help us detect where exactly the marble may be inside of Fluffy. The Fourier Transform represents the frequency signal in terms of *sines* and *cosines* (Gin). The function used to take the Fourier Transform of a function is:

$$F(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-ikx} f(x)dx$$ where f(x) is the time signal.

Similarly, there is an *inverse Fourier Transform* function which will convert a frequency signal back into a time signal. That function is:

$$f(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-ikx} F(x)dx$$ where F(k) is the frequency signal.

Both of these functions will be helpful in our problem, because we will be moving from the given time signal to obtain frequency measures of the marble and then back into the time signal to obtain where in time the marble's center frequency measures lie.

Because the measurements given to us contain noise, we also want to be able to filter this calculated frequency signal in order to determine a clearer path of the marble and hence know exactly where we need to focus on in order to break it up. The filtering that we will be using in this problem will be ***Gaussian Filtering.***

The Gaussian filter is a common filter that is used by multiplying the frequency signal (that we found out using the Fourier Transform of the time signal) and the Gaussian function:

$$\underline{\text{Gaussian Filter Function}} = e^{-\tau(K-k0)^2}$$

This function focuses on the given frequency k0 and normalizes the entire function (everything not important decays to 0). That rate that it focuses on the given frequency is based on τ. The Gaussian function is most commonly used because it gives us a simple and smooth version of the frequency signal graphs (Gin).

4. Algorithm Implementation and Development:

The method of finding the trajectory of the marble in Fluffy has been broken down into several steps, first finding the average frequency (calculating a center frequency from that), and then filtering the frequency signal, and finally converting it back to the time signal. All of this implementation is done in Matlab.

1. One of the first steps is to load in our data using the command
   load Testdata
2. Our problem (for the time signal) starts on the domain from L = -15 to 15 and use 64 points on each x, y, and z axis.
3. Using the following command, we are able to create the time signal Un for each of the 20 measurements in the data.
   Un(:,:,:)=reshape(Undata(jj,:),n,n,n);
4. We then create the frequency signal by taking the Fourier transform of the signal by using the built in Matlab command fftn. Making an emphasis that we are using fftn not fft because our data is in 3D. We also scale the time signal domain by $2\pi/2L$ to fit into the frequency domain to get -2π to 2π. We finally use meshgrid on both the time and frequency domains to get a grid for our plots. Putting these few steps together we get the following code:

```
L=15;
n=64;
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);
```

5. To average the frequencies, we first have to convert the 3d data to a frequency signal using fftn and then we get end up getting all of our frequency signals. We then average all of these frequencies to make the focus less on the random data and more on the frequent/more important data (leaving the frequencies that are more concentrated on the marble). We then normalize our data by dividing everything by the max of the average. This way we know our values will range from

```
ave = zeros(1,n);
for jj = 1:20
      Un(:,:,:)=reshape(Undata(jj,:),n,n,n);
      utn = fftn(Un);
      ave = ave + utn;
end
ave = abs(ave)/20;
ave_norm = ave./ max(ave,[],'all');
```

To find the center frequency, we know that the marble is going to have the largest part of the averaged signal, thus we need to check the maximum value of the averaged signal and at which x, y, and z frequencies it occurs at. We can calculate

this by seeing where the frequencies equal the max value of our average signal and get each kx, ky, and kz from that.

```
max_avg = max(ave,[],'all');
[kx_index, ky_index, kz_index] = ind2sub(size(ave), find(ave ==
            max_avg));
kx0 = Kx(kx_index, ky_index, kz_index);
ky0 = Ky(kx_index, ky_index, kz_index);
kz0 = Kz(kx_index, ky_index, kz_index);
```

6. We then have to filter the data around the calculated center frequency that we calculated by applying the Gaussian Filter at that frequency to eliminate any noise. Since our data was in 3D we end up having 3 different frequencies. Furthermore, the Gaussian function mentioned above will slightly be tweaked to fit our data. Instead of the square of the difference of one frequency (like in our function), we will sum the square of the differences of each (kx, ky, and kz) frequency and multiply that by Tau giving us the function: $e^{-\tau((Kx-kx0)^2+(Ky-ky0)^2+(Kz-kz0)^2}$

   filter = exp(-tau*((Kx-kx0).^2+(Ky-ky0).^2+(Kz-kz0).^2));

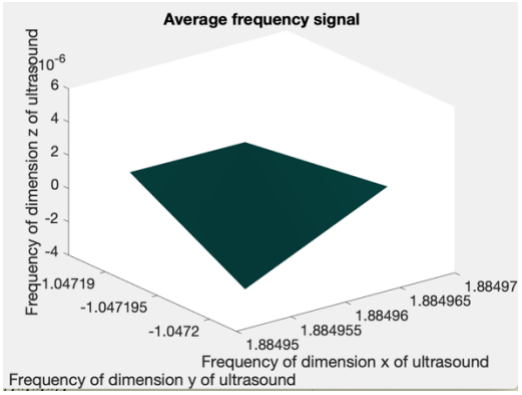
7. Now we apply the filter to our data by taking the filter function multiplied by each (fourier transformed) frequency signal and then converting that to (fftshifted-to align the domains of our data) a time signal by taking the inverse fourier transform (in 3d). This will give us the spatial variations of the marble (with the least random frequency) over time.

```
Un(:,:,:)=reshape(Undata(i,:),n,n,n);
unft = filter.*(fftn(Un));
Unf = ifftn(fftshift(unft));
```

5. Computational Results:

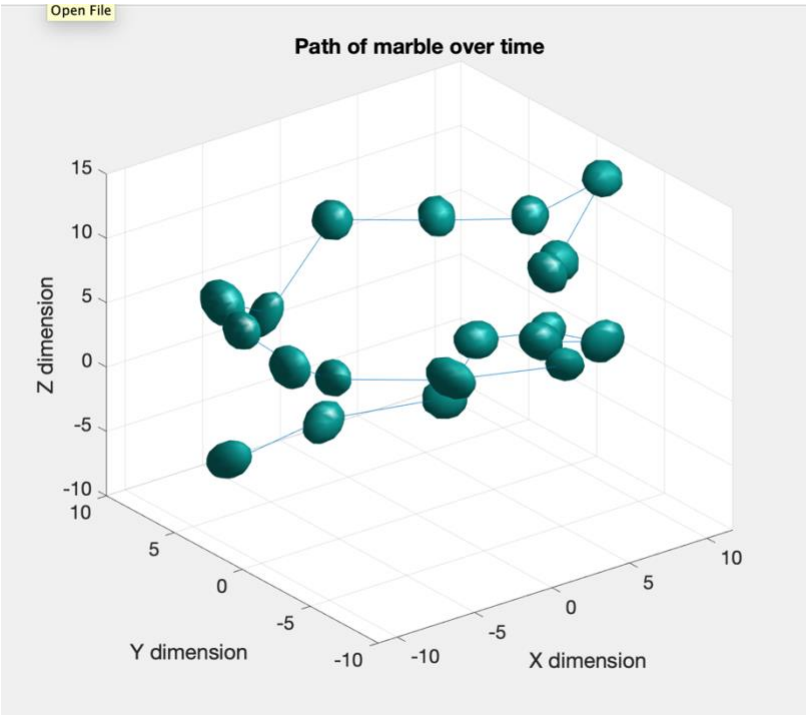
The average frequency signal (as seen below) that we created was able to help us in our overall goal by giving us the location of the center frequency. We were able to find the maximum value of the average signal to focus on where the marble's path would be which is exactly why computing the average signal was so important.

```
ave = zeros(1,n);
for jj = 1:20
       Un(:,:,:)=reshape(Undata(jj,:),n,n,n);
       utn = fftn(Un);
       ave = ave + utn;
end
ave = abs(ave)/20;
ave_norm = ave./ max(ave,[],'all');
isosurface(Kx, Ky, Kz, fftshift(ave_norm), .99999)
hold on
```



We can now track the path of the marble (as seen in the figure below) by plotting the location of the centroid (where we would want to break the marble) at each time. In our graph, the line connects the centroid of the marble at each time. We will keep a list of the coordinates of those locations in a matrix called path and thus will be able to see the marble's location as it moves through time.

```
path = zeros(20,3);
for i = 1:20
[index1, index2, index3] = ind2sub(size(Unf),find(abs(Unf) ==
max(abs(Unf),[],'all')));
path(i,1) = X(index1, index2, index3);
path(i,2) = Y(index1, index2, index3);
path(i,3) = Z(index1, index2, index3);
grid on;
hold on
end
plot3(path(:,1), path(:,2), path(:,3))
set(gca, 'Fontsize', 16)
xlabel('X dimension')
ylabel('Y dimension')
zlabel('Z dimension')
title('Path of marble over time')
```



6. Summary and Conclusions:

In conclusion the command path(20,:) gives us -7.9688, 4.6875, -5.6250 which are the x, y, and z coordinates (respectively) of where the acoustic wave should be performed at to break the marble in Fluffy. We were able to calculate these coordinate values by first Averaging, using Fourier Transforms and Inverse Fourier Transforms, and Gaussian filtering to pull out the most significant data from the noisy dataset.

7. Appendix A – Function Descriptions:

fftn: Fourier transform (in the data's dimension)
ifftn: Inverse Fourier Transform (in n-dimension)
fftshift: switching the first and second half of the domain so that it aligns when plotted
max(__, '[]', 'all'): finding the maximum most value in the *entire* array
abs: absolute value
linspace: create an array of n points, from for ex. -L to L
meshgrid: returns 2D coordinates based on the passed in vectors ex. (x,y)
exp: e raised to any power passed in
isosurface: connects points that have the specified isovalue with the given parameters passed in as the coordinate arrays
ind2sub: returns the multidimensional subscripts corresponding to what we are trying to 'find' from the passed in array.

8. Appendix B:

```
clear; close all; clc;load Testdata
    L=15; %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); %for time signal
    [Kx,Ky,Kz]=meshgrid(ks,ks,ks); %for frequency
    %  for j=1:20
    %     Un(:,:,:)=reshape(Undata(j,:),n,n,n);
    %     %close all, isosurface(X,Y,Z,abs(Un),0.4)
    %     %axis([-20 20 -20 20 -20 20]), grid on, drawnow
    %     %pause(1)
    %  end

    %PART 1:
    %creating an average of the spectrums:
    ave = zeros(1,n);
    for jj = 1:20
        Un(:,:,:)=reshape(Undata(jj,:),n,n,n);
        utn = fftn(Un);%take the fftn of each time signal->frequency
signal
        ave = ave + utn;%sum up frequencies
    end
    ave = abs(ave)/20; %because there are 20 measurements (take avg.
of freqs.)
    %use fftshift so that the values align on the plot
    ave_norm = ave./ max(ave,[],'all'); %scale it->normally using max
of 3d array
    %plot the average frequency, we know the isovalue because we
scaled all
    %values to be between 0-1
    isosurface(Kx, Ky, Kz, fftshift(ave_norm), .99999)
    hold on
    set(gca, 'Fontsize', 16)
    xlabel('Frequency of dimension x of ultrasound')
```

```
41.      ylabel('Frequency of dimension y of ultrasound')
42.      zlabel('Frequency of dimension z of ultrasound')
43.      title('Average frequency signal')
44.
45.      %%
46.      %Part 2/3:
47.      %find the largest point/center frequency
48.      max_avg = max(ave,[],'all');
49.      [kx_index, ky_index, kz_index] = ind2sub(size(ave),find(ave ==
   max_avg));%finding the indices of the max value of our avg freq signal
50.
51.      %filter the data around the center frequency signature(center
   frequency)
52.      tau = 20;
53.      kx0 = Kx(kx_index, ky_index, kz_index);%the frequency at that
   each index found
54.      ky0 = Ky(kx_index, ky_index, kz_index);
55.      kz0 = Kz(kx_index, ky_index, kz_index);
56.      filter = exp(-tau.*((Kx-kx0).^2+(Ky-ky0).^2+(Kz-kz0).^2));%3d
   gaussian filter
57.      path = zeros(20,3);
58.      for i = 1:20
59.          Un(:,:,:)=reshape(Undata(i,:),n,n,n);
60.          unft = filter.*(fftn(Un));%take the fttn of each time signal-
   >frequency
61.          Unf = ifftn(fftshift(unft));%applying the inverse of each
   filtered frequency signal->time signal
62.          isosurface(X, Y, Z,
   (abs(Unf)/max(abs(Unf),[],'all')),.99)%plotting each filtered time-
   signal
63.          [index1, index2, index3] = ind2sub(size(Unf),find(abs(Unf) ==
   max(abs(Unf),[],'all')));%finding the indices of the centroid of each
   marble
64.          path(i,1) = X(index1, index2, index3);%saving the X dimension
   of the centroid of the marble at each time
65.          path(i,2) = Y(index1, index2, index3);%saving the Y dim
66.          path(i,3) = Z(index1, index2, index3);%saving the Z dim
67.          grid on;
68.          hold on
69.      end
70.      plot3(path(:,1), path(:,2), path(:,3)) %plot the X,Y, and Z paths
   of the centroid of marble at each time
71.      set(gca, 'Fontsize', 16)
72.      xlabel('X dimension')
73.      ylabel('Y dimension')
74.      zlabel('Z dimension')
75.      title('Path of marble over time')
```

-(Gin): in class lecture/notes reference.