The Fourier Transform: An Ultrasound Problem

Alyssa Mell

Abstract

Using ultrasound measurements taken over time, we determine the location and trajectory of a marble within a dog's intestines. First, we average over the 20 time measurements in order to reduce noise in the data. Then, after taking the Fourier Transform, we find the center frequency of the averaged data and construct a filter to further reduce noise. Finally, we find the location of the marble at each of the 20 measurements. We use this information to determine where an intense acoustic wave should be focused in order to break apart the marble at the 20th data measurement.

Sec. I. Introduction and Overview

Ultrasounds have extremely useful applications in medicine, particularly with medical imaging. The frequencies observed in an ultrasound scan provide information about where a given mass is located within a body. However, the data coming directly from the ultrasound can contain a significant amount of noise. In such situations, the data must be manipulated in order to provide useful information.

In our particular case, a marble has been swallowed by a dog and it must be located using ultrasound so that it can be broken apart with an intense acoustic wave in order to save the dog's life. We have 20 sets of data taken at different times which provide us with measurements of spatial variations in a small volume of the dog's intestines where the marble is expected to be. Our goal is to find the location and trajectory of the marble within the dog's intestines. Unfortunately, internal fluid movement inside the dog's body causes noisy data that must be manipulated in order to accurately discern the location of the marble. By averaging and filtering our data measurements, and using the Fourier Transform to study the data in frequency space, we can eliminate noise in the data and compute an accurate location for the marble.

Sec. II. Theoretical Background

The main methods for data analysis used in this paper are averaging and filtering. The Fourier Transform is also a key factor in solving this problem, so it's important to understand how and why it works in our application.

If there is truly noise present in a given data set, then averaging over the time measurements should cancel it out because noise follows a roughly Gaussian distribution around each frequency value at a given point in frequency space. Averaging the data over multiple trials in frequency space therefore makes the central frequency more visible as noise is reduced. The central frequency is the maximum valued frequency that would be present in the absent of noise. Therefore, this is a useful method for determining the value of k_0, the central frequency value, for a filter.

There are many different types of filters that could be used, but a fairly simple one is the Gaussian filter. In its simplest form, this filter is as follows:

$$[1] \qquad f(k) = e^{-\tau(k-k_0)^2}$$

Where τ represents the filter width, and k_0 is the center frequency. The filter is a function of k, which represents all possible frequency values. Filters are very useful when trying to analyze noisy data in frequency space because, when applying the filter, the center frequency within the data is magnified. When this filtered data is then converted from frequency space back to position or time space, the original signal becomes much more clear.

This conversion between frequency space and time or position space is where the Fourier Transform is necessary. The Fourier Transform provides information about data in terms of its frequencies. It takes data that is given in time or position space and extracts all possible frequencies that make up that data. Then, in order to return the data to its original form, the inverse Fourier Transform can be taken. In MATLAB, this is denoted at the Fast Fourier Transform (FFT) which is simply an algorithm used to compute the Discrete Fourier Transform (DFT).

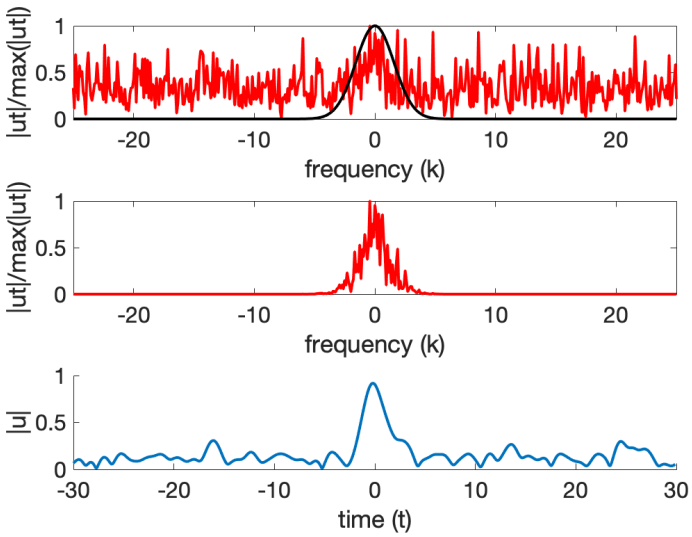

Figure 1: Top to bottom, this figure shows noisy data with an overlaid filter, the filtered data, and the resulting data after an inverse Fourier Transform (figure credit: Craig Gin).

Figure 1 shows a simplified 1D example of the problem that will be solved in this paper. First, a Fourier Transform of the data is taken in order to get to the frequency domain. This appears in the top plot along with an overlaid filter. Then, the appropriate filter is multiplied by the data in order to reduce noise around the center frequency. This is shown in the middle plot. Finally, the inverse Fourier Transform is taken in order to transform the data back to the time domain. This is shown in the bottom plot, where it is now clear that there is a signal other than noise present in the time domain. Our goal in this paper will be to follow this same general process, but in three dimensions, and with the added step of averaging our initial data measurements.

Sec. III. Algorithm Implementation and Development

In this section there will often be references to the MATLAB code which is located in Appendix B. The specific line of the code that is being referenced will be stated at the end of the sentence.

The first step in forming a solution in MATLAB was to create the arrays containing the position and frequency values for the data we have been given. The frequency values were re-scaled by 2π over the length of the space, $2L$, because the fast Fourier Transform (FFT) assumes that there will be 2π periodic signals (line 6). These position and frequency values were fit into 64^3 arrays in order to match the number of data measurements taken by the ultrasound (lines 8,9). The data provided for the 20 time measurements was also converted to a 64^3 array of values in order to mirror the position space over which the data was captured (line 12).

After initializing the data arrays for this problem, the next step was to average over all 20 measurements in order to reduce noise in the data. Figure 4 in Sec. IV. shows an isosurface plot of the initial noisy data at the first time measurement in frequency space. Using a `for` loop, the 20 data measurements were combined together and averaged. The Fourier Transform was taken at each time measurement in order to transform the data to frequency space (lines 23-28). Figure 5 in Sec. IV. shows the averaged data over the 20 trials in frequency space. In this figure it is clear that there is a maximum frequency value. This maximum value is what we want to use to filter our data because it corresponds to the center frequency. The MATLAB function ind2sub() was used to extract the indices of the maximum value (line 44). These indices could then be plugged into the frequency arrays $K_x, K_y,$ and K_z in order to get the center frequencies of the system in the x, y, and z directions (lines 47-49).

The next step was to build a filter using the center frequency values. Since there are three frequency values, one for each of the x, y, and z directions, all three of them need to be included in the filter, so an extended version of equation 1 is necessary (line 53). This leads to the following equation:

$$[2] \qquad f(k) = e^{-\tau((K_x-K_{x0})^2+(K_y-K_{y0})^2+(K_z-K_{z0})^2)}$$

Where K_x, K_y, K_z contain a grid of frequency values for each of three directions, and K_{x0}, K_{y0}, K_{z0} are the central frequency values. This equation was multiplied by the Fourier

Transform of the data at each time step, and then the inverse Fourier Transform was taken in order to transform the filtered data back to position space (lines 68,69). A visual depiction of this can be seen for each time step in Figure 6 of Sec. IV. Since we are back in position space, and most of the noise has been eliminated, it is clear from this graph that we are now looking at something in the shape of a marble. The value of τ was chosen to be 20 based on trial and error methods used to determine which value would produce the best marble shape on the plot. The command used to create this plot was isosurface(), which required an isovalue. The isosurface() function creates a plot such that all the data with the given isovalue shows up on the plot. In order to ensure that the correct isovalue was used, the data was normalized so that its maximum value would be one, and an isovalue slightly below one was chosen. Since the data's maximum value corresponds to its center on the isosurface plot, this led us to our next step - finding the center of the marble at each time. It was necessary to find these central values in order to draw a path connecting the marble at each of the 20 time steps and determine the marble's position at the 20th data measurement.

Consider the following plots with different isovalues. Figure 3 with the larger isovalue is more accurate, and closer to the center of the marble, than Figure 2 with the smaller isovalue. The center values of the marble at each point were found by using the indices of the max value of the data in position space. Plugging those indices into the position arrays X, Y, Z, gave the location of the marble at each point in time (lines 74-77). These values were stored in vectors and used as inputs for plot3() to show how the marble traveled over time (line 86). This is shown in Figure 7. Since we determined the center of the marble at each point in time, we were also able to answer our ultimate question of where the marble was located at the 20th measurement. This was done by extracting the data for the position of the point at time index 20 (line 94).

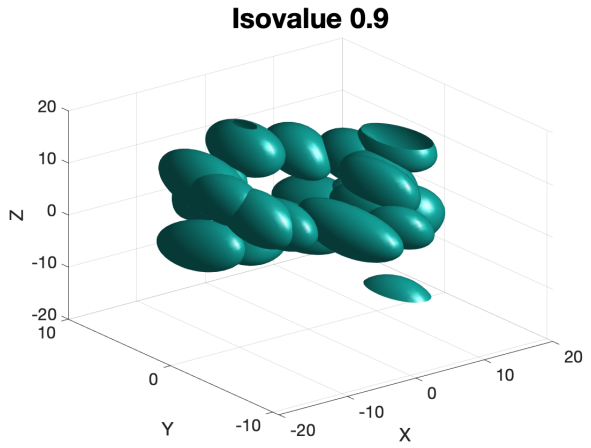

Figure 2: This plot shows the marble data plotted with isovalue 0.9.

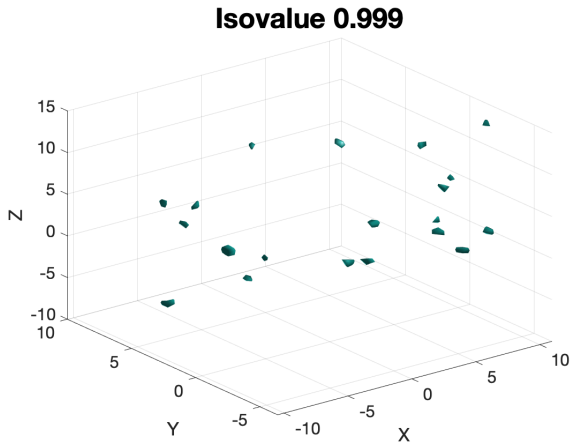

Figure 3: This plot shows the marble data plotted with isovalue 0.999.

Sec. IV. Computational Results

In the plots below, the computational results from the data analysis can be seen. The two plots in the first row show how noise was reduced in the data by averaging over the 20 measurements. The two plots in the second row show the filtered data in position space for the marble's location over time. Using the central value of the marble's location, we can determine that at time 20, it is located at the point $(-7.9688, 4.6875, -5.6250)$.

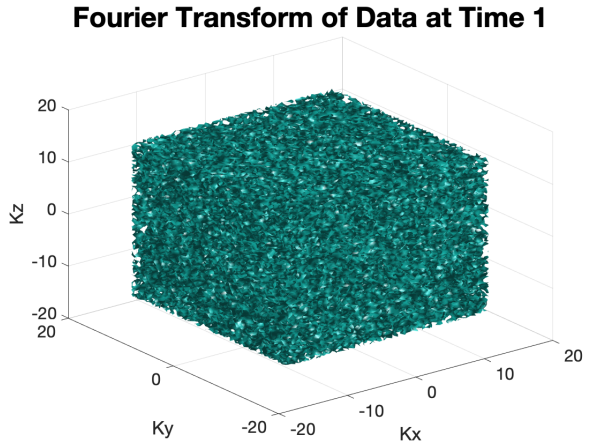

Figure 4: This plot shows the noisy data from measurements at time 1 in frequency space. Data from any other time measurement would produce a similar plot (isovalue = 0.4).

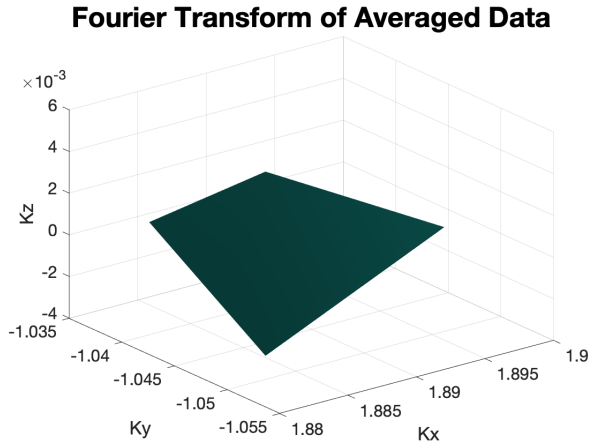

Figure 5: This plot shows the averaged data over all 20 data measurements in frequency space. Some of the noise has been reduced and the plot now has a clear maximum point (isovalue = 0.9).

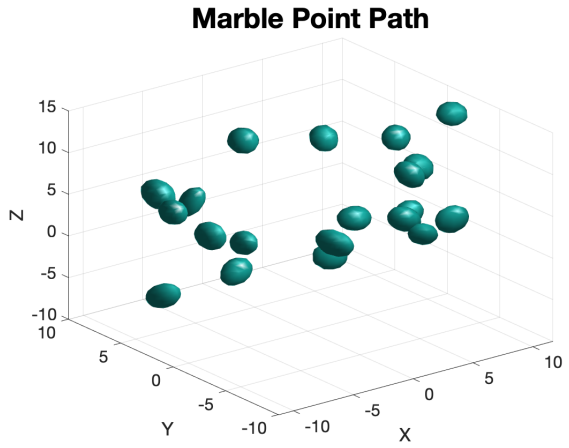

Figure 6: This plot shows the filtered data in position space at each of the 20 time measurements. There is a clear marble shape at each time measurement in the graph (isovalue = 0.99).

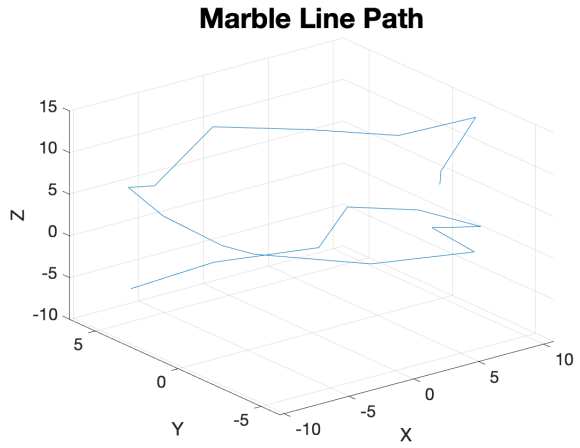

Figure 7: This plot shows the path of the marble over time in position space, starting at the top right corner. It connects the center of the figures shown in the plot to the left.

Sec. V. Summary and Conclusions

By using the Fourier Transform, along with methods such as averaging and filtering, we were able to reduce the noise in the ultrasound data and draw conclusions based on the resulting figures and values. From this manipulated data, it was possible to determine the location and trajectory of the marble within the dog's intestines. Using the location of the marble at the 20th data measurement, we can conclude that an intense acoustic wave should be focused at the point $(-7.9688, 4.6875, -5.6250)$ in order to break up the marble and save the dog's life.

Appendix A. MATLAB functions

Listed below are the important MATLAB functions used in this paper, along with a brief explanation of how they are implemented.

- `fftn`(X) : returns the discrete Fourier Transform of an N-dimensional array X.
- `fftshift`(X) : shifts the zero-frequency component to the center of the spectrum so that values will appear in the correct order when graphing.
- `ifftn`(X) : returns the inverse of the discrete Fourier Transform of an N-dimensional array X.
- [I,J,K] = `ind2sub`(size(X), find(X==n)) : returns the indices corresponding to the location of a given value n in array X. For our purposes, X is a 3D array so it returned three values.
- `isosurface`(X,Y,Z,V,ISOVALUE) : plots the array V with axes X, Y, and Z at the given ISOVALUE.
- `max`(X, [ ], 'all') : returns the largest element of X, where X is a multidimensional array.
- [X,Y,Z] = `meshgrid`(x,y,z) : creates a 3D rectangular grid (X,Y,Z) using x as the columns of X, y as the rows of Y, and z as the pages of Z.
- `plot3`(x,y,z) : plots a line in 3D space using the components of x, y, and z at each index to create each of the points.
- `reshape`(X,N,N,N) : converts the 1xM vector X into a 3D array with NxNxN dimensions.

Appendix B. MATLAB Codes

```
clear; close all; clc
load Testdata % load data from 20 measurements
L=15; % spatial domain
n=64; % fourier modes
x2=linspace(-L,L,n+1); x=x2(1:n); y=x; z=x; % evenly spaced
    positions values
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; % scaled frequency values
ks=fftshift(k); % shifted frequency values
[X,Y,Z]=meshgrid(x,y,z); % 3D grid of position values
[Kx,Ky,Kz]=meshgrid(ks,ks,ks); % 3D grid of frequency values

% Plots the data at time 1 before averaging and reducing noise.
Un(:,:,:)=reshape(Undata(1,:),n,n,n);
close all; isosurface(X,Y,Z,abs(Un),0.4)
grid on; drawnow
set(gca,'FontSize',15)
xlabel('Kx'); ylabel('Ky'); zlabel('Kz');
title('Fourier Transform of Data at Time 1', 'Fontsize', 25);
grid on; drawnow
print('time_1', '-dpng');

% The following code averages the 20 data measurements in Undata
    and
% uses fftn to convert them to frequency space.
avg = zeros(n,n,n);
for t=1:20
    Un(:,:,:) = reshape(Undata(t,:),n,n,n);
    avg = avg + fftn(Un);
end
avg = avg/20;

% Plots the averaged measurements from above in frequency space.
% In order to graph correctly, the data is shifted using fftshift
close all;
isosurface(Kx, Ky, Kz, fftshift(abs(avg))/max(abs(avg),[],'all')
    ,0.99)
set(gca,'FontSize',15)
xlabel('Kx'); ylabel('Ky'); zlabel('Kz');
title('Fourier Transform of Averaged Data', 'Fontsize', 25);
grid on; drawnow
print('average', '-dpng');

```

```
% Uses the maximum value of the averaged data set in order to find
% the indices of the center x,y, and z frequencies.
max_val = max(abs(avg),[],'all'); % find maximum value across all averaged data
a_avg = abs(avg);
[i_max,j_max,k_max] = ind2sub(size(a_avg),find(a_avg == max_val)); % indices of max value

% Uses indices from above to set filter center.
k0_x = Kx(i_max,j_max,k_max); % x frequency
k0_y = Ky(i_max,j_max,k_max); % y frequency
k0_z = Kz(i_max,j_max,k_max); % z frequency

tau = 20; % filter width

filter = exp(-tau.*((Kx-k0_x).^2+(Ky-k0_y).^2+(Kz-k0_z).^2)); % create filter

% Initiazes vectors to hold values for location of marble at each time.
x_path = zeros(1,20);
y_path = zeros(1,20);
z_path = zeros(1,20);

% Loop goes through each of the 20 data measurements. Applies filter each
% time and plots the filtered data with reduced noise using isosurface.
% Also determines the center of the data at each measurement and saves
% those x, y, and z values in the vectors created above to mark the path of
% the marble.
close all;
for t = 1:20
    Un(:,:,:) = reshape(Undata(t,:),n,n,n);
    Un = filter.*fftn(Un);
    Un = ifftn(Un);
    isosurface(X,Y,Z,abs(Un)/max(abs(Un),[],'all'),0.99)
    grid on; drawnow
    hold on
    max_val = max(abs(Un),[],'all');
    [i_max,j_max,k_max] = ind2sub(size(Un),find(abs(Un) == max_val));
    x_path(t) = X(i_max,j_max,k_max);
```

```
76      y_path(t) = Y(i_max,j_max,k_max);
77      z_path(t) = Z(i_max,j_max,k_max);
78  end
79  set(gca,'FontSize',15)
80  xlabel('X'); ylabel('Y'); zlabel('Z');
81  title('Marble Point Path', 'Fontsize', 25);
82  print('iso_path', '-dpng');
83
84  % Plots the path of the marble over time.
85  close all;
86  plot3(x_path,y_path,z_path);
87  set(gca,'FontSize',15)
88  xlabel('X'); ylabel('Y'); zlabel('Z');
89  title('Marble Line Path', 'Fontsize', 25);
90  grid on; drawnow
91  print('path', '-dpng');
92
93  % Saves location of marble at 20th time measurement.
94  final_location = [x_path(20), y_path(20), z_path(20)];
```